Azure Data Factory Notes:

Adf Interview Question:

-----M3BI (Zensar compnay)------
1. Write a SQL query of percentage calculation for growth and losses in year?
2. If a table has both positive and negative values. How do we differentiate them using SQL query?
3. If a file is stored in a local machine and storage account. What is the difference between them?
4. How frequently do we you excute ADF pipeline.
5. How long do you excute pipeline?
6. Do you have idea on Dataware house concepts?
7. What is your sources?
8. Difference between azure cloud and other cloud?
9. What is performance tuning?

-----persistant------
1. Difference between functions and procedures?
2. what are ADF activities and explain?
3. How to move data from one storage to another storage?
4. What are YAML files and azure devops?
5. Why we cannot use the functions in parameters?

------LTI Mindtree-----
1. If a file size is greater than 5MB that has to be moved from source and destination. what are the activities used to create pipeline. what is the input and output of each activity?
2. Difference between azure blob and ADLSGen2?
3. How the parquet files look like?
4. Differnce between delete, drop and truncate?
5. Write a SQL query that has both positive values and negative values. How we can seperate them?
6. How do we move pipeline from development stage to production environment?
7. What are SAAS and PAAS tokens?
8. What are different types of triggers?
9. Whata are your roles and responsibilities?

-----TEK Syatems-----
1. What are relationships in SQL?
2. Write a SQL query that has columns employee,name and salary. Third highest salary will be displayed?
3. What are the different ways to execute pipeline?
4. How do we move pipeline from development stage to production environment?
5. Explain the incremental load pipeline?
6. What are triggers?
7. What is integration runtime and it types?

-----Lakshya Software (Atos Syntel)-----
1. How do you cout files in one storage account?
2. There are 4 files, How do we copy last 2 files from source to destination?
3. 5. Explain the incremental load pipeline?

-----Sonata Software-----
1. What are different activites and explain each?
2. If a file is 1gb, how can we move data from source to destintion?
3. Differnce between Azure SQL and MS SQL?
4. Differecne between azure blob and ADLSGen2?
5. What are disadvantages of ADF?
6. How do we moniter a pipeline?
7. What are triggers and types?
8. What is logic apps?
9. What are relationship in SQL?
10. What are joins and types?
11. Why do we write a store procedure in ADF and SQL?
12. What is normalization and cursors in SQL?
13. What is difference between primary key, unique key and surrogate key?
14. Diference between dataware house and ADLSGen2?
15. What is use of pipeline?
16. How a powerbi guy pull a pipeline from ADF?

-----Abcu services (Infosys)-----
1. Diference between dataware house and ADLSGen2?
2. What are default values in parametres?
3. What are different activities in ADF?
4. What are triggers?
5. Copy behaviour in ADF?

-----Wipro----
1. What are variables and parametres in Azure?
2. How to create generic pipeline based on source. The source must be .csv, xml, parquet, json?
3. Through SQL DB, How we will generte a parqust file using ADF?
4. How do we connect to SQL DB?
5. What is incremantal loa and full load?
6. Difference between debug an trigger now?
7. Based on last two data that has to be copied from source to destination?
8. What are ways to execute a pipeline?
9. What is integration run time?
10. Difference lookup and metadata activity?

-----BPKtech-----
1. How do we connect our ADF to on premise?
2. How to improve the performance of copy activity?
3. Explain incremantal load?
4. Write a SQL query to calculate balance?
5. Explain the procedure to connect the on premise data?
6. When we use the stagging area in copy activity?

-----infogain-----
1. Explain incremental load?
2. What is IR and it types?
3. What is mapping data flows?
4. What is rowset number in synapse?
5. What are nodes in azure synapse?

-----TechM-----
1. How to move pipeline in ADF (not copy)?
2. Earlier a store produre in taking 5 minutes to excute and now it is taking 30 mins why?
3. Triggers (Tumbling window)?
4. What is the difficulty you faced while using ADF?
5. Table: Employee

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| salary      | int     |
| managerId   | int     |
+-------------+---------+
```

id is the primary key column for this table. Each row of this table indicates the ID of an employee, their name, salary, and the ID of their manager.

Write an SQL query to find the employees who earn more than their managers. Return the result table in any order. The query result format is in the following example.

Example 1:

Input:
Employee table:

```
+----+-------+--------+-----------+
| id | name  | salary | managerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | Null      |
| 4  | Max   | 90000  | Null      |
+----+-------+--------+-----------+
```

Poly Base Mechanism Adf:

Create master key encryption by password='Appuro@10111'

```
CREATE DATABASE SCOPED CREDENTIAL AZURESTORAGECREDENTIAL
 WITH
  IDENTITY='RAM',
  secret='6v/l4TkuJv7W88FUNt74tIBC0Em1ty6Cn6ZtWx2GLxF3MOkMdQgek6WLaWPN79yV5Axi7ZfTfWLG+AStss5i6g=='

CREATE EXTERNAL DATA SOURCE AZURESTORAGE
 WITH
  (
```

```
        TYPE=HADOOP,---blob or datalake
        LOCATION='wasbs://employee@eclasessmemorylake.blob.core.windows.net',
        CREDENTIAL=AZURESTORAGECREDENTIAL

      )


CREATE EXTERNAL FILE FORMAT EMPFILE
 WITH
   (  FORMAT_TYPE=delimitedtext,
     format_options(field_terminator=',',FIRST_ROW = 2)
   )

CREATE EXTERNAL TABLE extemp
(
 eno INT,
 ename VARCHAR(100),
 sal INT
)
 WITH
  (DATA_SOURCE=AZURESTORAGE,
  FILE_FORMAT=EMPFILE,
  LOCATION='/emp/'
  )

select * from extemp



CTAS--create table as select

CREATE Table Dimemp
with (distribution=round_robin)
As select * from extemp

SELECT * FROM Dimemp
```

Azure Data Bricks:

Explanation of all PySpark RDD, DataFrame and SQL examples present on this project are available at Apache PySpark Tutorial, All these examples are coded in Python language and tested in our development environment.

Table of Contents (Spark Examples in Python)

PySpark Basic Examples:

How to create SparkSession
PySpark – Accumulator
PySpark Repartition vs Coalesce
PySpark Broadcast variables
PySpark – repartition() vs coalesce()
PySpark – Parallelize
PySpark – RDD
PySpark – Web/Application UI
PySpark – SparkSession
PySpark – Cluster Managers
PySpark – Install on Windows
PySpark – Modules & Packages
PySpark – Advantages
PySpark – Features
PySpark – What is it? & Who uses it?
PySpark DataFrame Examples
PySpark – Create a DataFrame
PySpark – Create an empty DataFrame
PySpark – Convert RDD to DataFrame
PySpark – Convert DataFrame to Pandas
PySpark – StructType & StructField
PySpark Row using on DataFrame and RDD
Select columns from PySpark DataFrame
PySpark Collect() – Retrieve data from DataFrame
PySpark withColumn to update or add a column
PySpark using where filter function
PySpark – Distinct to drop duplicate rows
PySpark orderBy() and sort() explained

PySpark Groupby Explained with Example
PySpark Join Types Explained with Examples
PySpark Union and UnionAll Explained
PySpark UDF (User Defined Function)
PySpark flatMap() Transformation
PySpark map Transformation
PySpark SQL Functions
PySpark Aggregate Functions with Examples
PySpark Window Functions
PySpark Datasources
PySpark Read CSV file into DataFrame
PySpark read and write Parquet File
==========
What is SparkSession

SparkSession was introduced in version 2.0,
It is an entry point to underlying PySpark functionality in order to
programmatically create PySpark RDD,DataFrame.
It's object spark is default available in pyspark-shell
and it can be created programmatically using SparkSession.

Spark Session:
---
pyspark.import sparksession
sparksession.builder

SparkSession also includes all the APIs available in different contexts:

SparkContext
SQLContext
StreamingContext
HiveContext.

How many SparkSessions can you create in a PySpark application?
sparksession()
SparkSession.builder()
SparkSession.newSession()

Usage of spark object in PySpark shell
>>>spark.version
3.1.2

Create SparkSession from builder?

```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[1]") \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()
```

Note: SparkSession object spark is by default available in the PySpark shell.

```
# Create new SparkSession
spark2 = SparkSession.newSession
print(spark2)

# Get Existing SparkSession
spark3 = SparkSession.builder.getOrCreate
print(spark3)

# Usage of config()
spark = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .config("spark.some.config.option", "config-value") \
    .getOrCreate()

# Enabling Hive to use in Spark
spark = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .config("spark.sql.warehouse.dir", "<path>/spark-warehouse") \
    .enableHiveSupport() \
    .getOrCreate()

# Set Config
spark.conf.set("spark.executor.memory", "5g")
```

```
# Get a Spark Config
partitions = spark.conf.get("spark.sql.shuffle.partitions")
print(partitions)

# Create DataFrame
df = spark.createDataFrame([("Scala", 25000), ("Spark", 35000), ("PHP", 21000)])
df.show()

# Output
#+-----+-----+
#|   _1|   _2|
#+-----+-----+
#|Scala|25000|
#|Spark|35000|
#|  PHP|21000|
#+-----+-----+

# Spark SQL
df.createOrReplaceTempView("sample_table") or we used createGlobalTempView()
df2 = spark.sql("SELECT _1,_2 FROM sample_table")
df2.show()

# Create Hive table & query it.
spark.table("sample_table").write.saveAsTable("sample_hive_table")
df3 = spark.sql("SELECT _1,_2 FROM sample_hive_table")
df3.show()
```

Working with Catalogs

To get the catalog metadata, PySpark Session exposes catalog variable.
spark.catalog.listDatabases()
spark.catalog.listTables()
returns the DataSet.

```
# Get metadata from the Catalog
# List databases
dbs = spark.catalog.listDatabases()
print(dbs)

# Output
#[Database(name='default', description='default database',
#locationUri='file:/Users/admin/.spyder-py3/spark-warehouse')]

# List Tables
tbls = spark.catalog.listTables()
print(tbls)

#Output
#[Table(name='sample_hive_table', database='default',
description=None, table, isTemporary=True)]
```

SparkSession Commonly Used Methods:

version()

createDataFrame() – This creates a DataFrame from a collection and an RDD

getActiveSession() – returns an active Spark session.

read()
readStream()
sparkContext() – Returns a SparkContext.

sql() – Returns a DataFrame after executing the SQL mentioned.

sqlContext() – Returns SQLContext.

stop() – Stop the current SparkContext.

table() – Returns a DataFrame of a table or view.

udf() – Creates a PySpark UDF to use it on DataFrame, Dataset, and SQL

What is PySpark Accumulator?

Accumulators are write-only and initialize once variables where only tasks that are running on workers are allowed to update and updates from the workers get propagated automatically to the driver program.

Accumulator variable using the value property.

How to create Accumulator variable in PySpark?

sparkContext.accumulator() > define accumulator variables.

add() > add/update a value in accumulator

value property on the accumulator variable is used to retrieve the value from the accumulator.
We can create Accumulators in PySpark for primitive types int and float.

PySpark accumulator examples:

```
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName("accumulator").getOrCreate()

accum=spark.sparkContext.accumulator(0)
rdd=spark.sparkContext.parallelize([1,2,3,4,5])
rdd.foreach(lambda x:accum.add(x))
print(accum.value)

accuSum=spark.sparkContext.accumulator(0)
def countFun(x):
  global accuSum
  accuSum+=x
rdd.foreach(countFun)
print(accuSum.value)

accumCount=spark.sparkContext.accumulator(0)
rdd2=spark.sparkContext.parallelize([1,2,3,4,5])
rdd2.foreach(lambda x:accumCount.add(1))
print(accumCount.value)
```

Complete Example of PySpark RDD repartition and coalesce:

```
# Complete Example
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com') \
      .master("local[5]").getOrCreate()

df = spark.range(0,20)
print(df.rdd.getNumPartitions())

spark.conf.set("spark.sql.shuffle.partitions", "500")

rdd = spark.sparkContext.parallelize(range(0,20))
print("From local[5]"+str(rdd.getNumPartitions()))

rdd1 = spark.sparkContext.parallelize(range(0,25), 6)
print("parallelize : "+str(rdd1.getNumPartitions()))

"""rddFromFile = spark.sparkContext.textFile("src/main/resources/test.txt",10)
print("TextFile : "+str(rddFromFile.getNumPartitions())) """

rdd1.saveAsTextFile("c://tmp/partition2")

rdd2 = rdd1.repartition(4)
print("Repartition size : "+str(rdd2.getNumPartitions()))
rdd2.saveAsTextFile("c://tmp/re-partition2")

rdd3 = rdd1.coalesce(4)
print("Repartition size : "+str(rdd3.getNumPartitions()))
rdd3.saveAsTextFile("c:/tmp/coalesce2")

# DataFrame example
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com') \
      .master("local[5]").getOrCreate()

df=spark.range(0,20)
print(df.rdd.getNumPartitions())

df.write.mode("overwrite").csv("c:/tmp/partition.csv")

# DataFrame repartition
```

```
df2 = df.repartition(6)
print(df2.rdd.getNumPartitions())

# Output:
Partition 1 : 14 1 5
Partition 2 : 4 16 15
Partition 3 : 8 3 18
Partition 4 : 12 2 19
Partition 5 : 6 17 7 0
Partition 6 : 9 10 11 13

# DataFrame coalesce
df3 = df.coalesce(2)
print(df3.rdd.getNumPartitions())

# Output:
Partition 1 : 0 1 2 3 8 9 10 11
Partition 2 : 4 5 6 7 12 13 14 15 16 17 18 19
```

---

spark.sql.shuffle.partitions

Union,groupBy and joins etc.

```
# Default shuffle partition count

df4 = df.groupBy("id").count()
print(df4.rdd.getNumPartitions())
```

Broad cast Variables:

```
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

states = {"NY":"New York", "CA":"California", "FL":"Florida"}
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),
   ("Michael","Rose","USA","NY"),
   ("Robert","Williams","USA","CA"),
   ("Maria","Jones","USA","FL")
 ]

columns = ["firstname","lastname","country","state"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

def state_convert(code):
   return broadcastStates.value[code]

result = df.rdd.map(lambda x: (x[0],x[1],x[2],state_convert(x[3]))).toDF(columns)
result.show(truncate=False)
```

Empty data parallelize etc:

```
emptyRDD = sparkContext.emptyRDD()
emptyRDD2 = rdd=sparkContext.parallelize([])

print("is Empty RDD : "+str(emptyRDD2.isEmpty()))
```

What is RDD (Resilient Distributed Dataset)?

RDD (Resilient Distributed Dataset) is a fundamental building block of PySpark which is fault-tolerant,
immutable distributed collections of objects. Immutable meaning once you create an RDD you cannot change it.
Each record in RDD is divided into logical partitions,
which can be computed on different nodes of the cluster.

map,filter,persist,groupByKey,join()

Note: RDD's can have a name and unique identifier (id)

PySpark RDD Benefits:

>In-Memory Processing
>immutable

>fault tolerant
>Lazy Evolution
>Partitioning

Create RDD:

```
from pyspark.sql import SparkSession
spark:SparkSession = SparkSession.builder()
    .master("local[1]")
    .appName("SparkByExamples.com")
    .getOrCreate()
```

master() – If you are running it on the cluster you need to use your master name
or
mesos depends on your cluster setup

appName() – Used to set your application name.

getOrCreate() – This returns a SparkSession object if already exists,
and creates a new one if not exist.

python
1 1,2 partition 1
2----> RDD CREATION -->
3 3,4 partition 2
4

```
#Create RDD from parallelize
data = [1,2,3,4,5,6,7,8,9,10,11,12]
rdd=spark.sparkContext.parallelize(data)
```

we mostly create RDD by using external storage systems like HDFS, S3, HBase e.t.c

sparkContext.txtfile:

```
#Create RDD from external Data source
rdd2 = spark.sparkContext.textFile("/path/textFile.txt")
```

sparkContext.Wholetextfile:

```
#Reads entire file into a RDD as single record.
rdd3 = spark.sparkContext.wholeTextFiles("/path/textFile.txt")
```

sparkContext.empty RDD:

```
# Creates empty RDD with no partition
rdd = spark.sparkContext.emptyRDD
# rddString = spark.sparkContext.emptyRDD[String]


#Create empty RDD with partition
rdd2 = spark.sparkContext.parallelize([],10) #This creates 10 partitions
```

RDD Parallelize:

parallelize() or
textFile() or
wholeTextFiles()
methods of SparkContxt to initiate RDD.

getNumPartitions() – This a RDD function which returns a number
of partitions our dataset split into.

```
print("initial partition count:"+str(rdd.getNumPartitions()))
#Outputs: initial partition count:2
```

Set parallelize manually:

```
sparkContext.parallelize([1,2,3,4,56,7,8,9,12,3], 10)
```

Repartition and Coalesce:

```
reparRdd = rdd.repartition(4)
print("re-partition count:"+str(reparRdd.getNumPartitions()))

#Outputs: "re-partition count:4
```

Note: repartition() or coalesce() methods also returns a new RDD.

PySpark RDD Operations:

RDD transformations – Transformations are lazy operations, instead of updating an RDD,
these operations return another RDD.

RDD actions – operations that trigger computation and return RDD values.

flatMap:
```
rdd2 = rdd.flatMap(lambda x: x.split(" "))
```

map:
```
rdd3 = rdd2.map(lambda x: (x,1))
```

reduceByKey:
```
rdd4 = rdd3.reduceByKey(lambda a,b: a+b)
```

sortByKey:

```
rdd5 = rdd4.map(lambda x: (x[1],x[0])).sortByKey()
#Print rdd5 result to console
print(rdd5.collect())
```

filter:
```
rdd4 = rdd3.filter(lambda x : 'an' in x[1])
print(rdd4.collect())
```

RDD Actions with example:

count() – Returns the number of records in an RDD

```
# Action - count
print("Count : "+str(rdd6.count()))
```
first() – Returns the first record.

```
# Action - first
firstRec = rdd6.first()
print("First Record : "+str(firstRec[0]) + ","+ firstRec[1])
```
max() – Returns max record.

```
# Action - max
datMax = rdd6.max()
print("Max Record : "+str(datMax[0]) + ","+ datMax[1])
```
reduce() – Reduces the records to single, we can use this to count or sum.

```
# Action - reduce
totalWordCount = rdd6.reduce(lambda a,b: (a[0]+b[0],a[1]))
print("dataReduce Record : "+str(totalWordCount[0]))
```
take() – Returns the record specified as an argument.

```
# Action - take
data3 = rdd6.take(3)
for f in data3:
    print("data3 Key:"+ str(f[0]) +", Value:"+f[1])
```
collect() – Returns all data from RDD as an array. Be careful when you use this action when you are working with huge RDD with millions and billions of data as you may run out of memory on the driver.

```
# Action - collect
data = rdd6.collect()
for f in data:
    print("Key:"+ str(f[0]) +", Value:"+f[1])
```
saveAsTextFile() – Using saveAsTestFile action, we can write the RDD to a text file.

```
rdd6.saveAsTextFile("/tmp/wordCount")
```

Types of RDD:

PairRDDFunctions or PairRDD – Pair RDD is a key-value pair This is mostly used RDD type

ShuffledRDD
DoubleRDD
SequenceFileRDD
HadoopRDD
ParallelCollectionRDD

ShuffledRDD:

repartition()
coalesce()
groupByKey()
reduceByKey()
cogroup() and join() but not countByKey().

RDD Persistant:
cache() and persist() etc.

Advantages of Persisting RDD:

Cost efficient – PySpark computations are very expensive hence reusing the computations are used to save cost.
Time efficient – Reusing the repeated computations saves lots of time.
Execution time – Saves execution time of the job which allows us to perform more jobs on the same cluster.

cachedRdd = rdd.cache().

RDD Persist method:

PySpark persist() method is used to store the RDD to one of the storage levels
MEMORY_ONLY,
MEMORY_AND_DISK,
MEMORY_ONLY_SER,
MEMORY_AND_DISK_SER,
DISK_ONLY,
MEMORY_ONLY_2,
MEMORY_AND_DISK_2

```
import pyspark
dfPersist = rdd.persist(pyspark.StorageLevel.MEMORY_ONLY)
dfPersist.show(false)
```

Unpersist:remove the data using by itselfs.

```
rddPersist2 = rddPersist.unpersist()
```

==========

Table of Contents (Spark Examples in Python):

UDF:USER DEFINED Function:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf
from pyspark.sql.types import StringType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

columns = ["Seqno","Name"]
data = [("1", "john jones"),
   ("2", "tracey smith"),
   ("3", "amy sanders")]

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

df.show(truncate=False)

def convertCase(str):
   resStr=""
   arr = str.split(" ")
   for x in arr:
      resStr= resStr + x[0:1].upper() + x[1:len(x)] + " "
   return resStr

""" Converting function to UDF """
convertUDF = udf(lambda z: convertCase(z))

df.select(col("Seqno"), \
   convertUDF(col("Name")).alias("Name") ) \
.show(truncate=False)

def upperCase(str):
   return str.upper()
```

```
upperCaseUDF = udf(lambda z:upperCase(z),StringType())

df.withColumn("Cureated Name", upperCaseUDF(col("Name"))) \
.show(truncate=False)

""" Using UDF on SQL """
spark.udf.register("convertUDF", convertCase,StringType())
df.createOrReplaceTempView("NAME_TABLE")
spark.sql("select Seqno, convertUDF(Name) as Name from NAME_TABLE") \
    .show(truncate=False)

spark.sql("select Seqno, convertUDF(Name) as Name from NAME_TABLE " + \
        "where Name is not null and convertUDF(Name) like '%John%'") \
    .show(truncate=False)

""" null check """

columns = ["Seqno","Name"]
data = [("1", "john jones"),
   ("2", "tracey smith"),
   ("3", "amy sanders"),
   ('4',None)]

df2 = spark.createDataFrame(data=data,schema=columns)
df2.show(truncate=False)
df2.createOrReplaceTempView("NAME_TABLE2")

spark.udf.register("_nullsafeUDF", lambda str: convertCase(str) if not str is None else "" , StringType())

spark.sql("select _nullsafeUDF(Name) from NAME_TABLE2") \
    .show(truncate=False)

spark.sql("select Seqno, _nullsafeUDF(Name) as Name from NAME_TABLE2 " + \
        " where Name is not null and _nullsafeUDF(Name) like '%John%'") \
    .show(truncate=False)

---------------------------------------------------------------------------------
         FlatMap Examples:
         -----------------
         from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = ["Project Gutenberg’s",
      "Alice’s Adventures in Wonderland",
      "Project Gutenberg’s",
      "Adventures in Wonderland",
      "Project Gutenberg’s"]
rdd=spark.sparkContext.parallelize(data)
for element in rdd.collect():
   print(element)

#Flatmap
rdd2=rdd.flatMap(lambda x: x.split(" "))
for element in rdd2.collect():
   print(element)
=================================================================================================
Map() examples:
---------------

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = ["Project",
"Gutenberg’s",
"Alice’s",
"Adventures",
"in",
"Wonderland",
"Project",
"Gutenberg’s",
"Adventures",
"in",
"Wonderland",
"Project",
"Gutenberg’s"]

rdd=spark.sparkContext.parallelize(data)
```

```
rdd2=rdd.map(lambda x: (x,1))
for element in rdd2.collect():
   print(element)

data = [('James','Smith','M',30),
  ('Anna','Rose','F',41),
  ('Robert','Williams','M',62),
]

columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()

rdd2=df.rdd.map(lambda x:
   (x[0]+","+x[1],x[2],x[3]*2)
   )
df2=rdd2.toDF(["name","gender","new_salary"]   )
df2.show()

#Referring Column Names
rdd2=df.rdd.map(lambda x:
   (x["firstname"]+","+x["lastname"],x["gender"],x["salary"]*2)
   )

#Referring Column Names
rdd2=df.rdd.map(lambda x:
   (x.firstname+","+x.lastname,x.gender,x.salary*2)
   )

def func1(x):
   firstName=x.firstname
   lastName=x.lastname
   name=firstName+","+lastName
   gender=x.gender.lower()
   salary=x.salary*2
   return (name,gender,salary)

rdd2=df.rdd.map(lambda x: func1(x))
================================================================================================
PySpark Aggregate examples:
---------------------------

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import approx_count_distinct,collect_list
from pyspark.sql.functions import collect_set,sum,avg,max,countDistinct,count
from pyspark.sql.functions import first, last, kurtosis, min, mean, skewness
from pyspark.sql.functions import stddev, stddev_samp, stddev_pop, sumDistinct
from pyspark.sql.functions import variance,var_samp,  var_pop

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James", "Sales", 3000),
   ("Michael", "Sales", 4600),
   ("Robert", "Sales", 4100),
   ("Maria", "Finance", 3000),
   ("James", "Sales", 3000),
   ("Scott", "Finance", 3300),
   ("Jen", "Finance", 3900),
   ("Jeff", "Marketing", 3000),
   ("Kumar", "Marketing", 2000),
   ("Saif", "Sales", 4100)
 ]
schema = ["employee_name", "department", "salary"]

df = spark.createDataFrame(data=simpleData, schema = schema)
df.printSchema()
df.show(truncate=False)

print("approx_count_distinct: " + \
     str(df.select(approx_count_distinct("salary")).collect()[0][0]))

print("avg: " + str(df.select(avg("salary")).collect()[0][0]))

df.select(collect_list("salary")).show(truncate=False)

df.select(collect_set("salary")).show(truncate=False)
```

```
df2 = df.select(countDistinct("department", "salary"))
df2.show(truncate=False)
print("Distinct Count of Department & Salary: "+str(df2.collect()[0][0]))

print("count: "+str(df.select(count("salary")).collect()[0]))
df.select(first("salary")).show(truncate=False)
df.select(last("salary")).show(truncate=False)
df.select(kurtosis("salary")).show(truncate=False)
df.select(max("salary")).show(truncate=False)
df.select(min("salary")).show(truncate=False)
df.select(mean("salary")).show(truncate=False)
df.select(skewness("salary")).show(truncate=False)
df.select(stddev("salary"), stddev_samp("salary"), \
   stddev_pop("salary")).show(truncate=False)
df.select(sum("salary")).show(truncate=False)
df.select(sumDistinct("salary")).show(truncate=False)
df.select(variance("salary"),var_samp("salary"),var_pop("salary")) \
 .show(truncate=False)
```

```
================================================================================
```

Source Code of Window Functions Example:

```
-----------------------------------------

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = (("James", "Sales", 3000), \
   ("Michael", "Sales", 4600),  \
   ("Robert", "Sales", 4100),   \
   ("Maria", "Finance", 3000),  \
   ("James", "Sales", 3000),    \
   ("Scott", "Finance", 3300),  \
   ("Jen", "Finance", 3900),    \
   ("Jeff", "Marketing", 3000), \
   ("Kumar", "Marketing", 2000),\
   ("Saif", "Sales", 4100) \
 )

columns= ["employee_name", "department", "salary"]

df = spark.createDataFrame(data = simpleData, schema = columns)

df.printSchema()
df.show(truncate=False)

from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
windowSpec  = Window.partitionBy("department").orderBy("salary")

df.withColumn("row_number",row_number().over(windowSpec)) \
   .show(truncate=False)

from pyspark.sql.functions import rank
df.withColumn("rank",rank().over(windowSpec)) \
   .show()

from pyspark.sql.functions import dense_rank
df.withColumn("dense_rank",dense_rank().over(windowSpec)) \
   .show()

from pyspark.sql.functions import percent_rank
df.withColumn("percent_rank",percent_rank().over(windowSpec)) \
   .show()

from pyspark.sql.functions import ntile
df.withColumn("ntile",ntile(2).over(windowSpec)) \
   .show()

from pyspark.sql.functions import cume_dist
df.withColumn("cume_dist",cume_dist().over(windowSpec)) \
  .show()

from pyspark.sql.functions import lag
df.withColumn("lag",lag("salary",2).over(windowSpec)) \
    .show()
```

```
from pyspark.sql.functions import lead
df.withColumn("lead",lead("salary",2).over(windowSpec)) \
   .show()

windowSpecAgg  = Window.partitionBy("department")
from pyspark.sql.functions import col,avg,sum,min,max,row_number
df.withColumn("row",row_number().over(windowSpec)) \
  .withColumn("avg", avg(col("salary")).over(windowSpecAgg)) \
  .withColumn("sum", sum(col("salary")).over(windowSpecAgg)) \
  .withColumn("min", min(col("salary")).over(windowSpecAgg)) \
  .withColumn("max", max(col("salary")).over(windowSpecAgg)) \
  .where(col("row")==1).select("department","avg","sum","min","max") \
  .show()
 ================================================================================
  PySpark Read CSV Complete Example

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
from pyspark.sql.types import ArrayType, DoubleType, BooleanType
from pyspark.sql.functions import col,array_contains

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

df = spark.read.csv("/tmp/resources/zipcodes.csv")

df.printSchema()

df2 = spark.read.option("header",True) \
    .csv("/tmp/resources/zipcodes.csv")
df2.printSchema()

df3 = spark.read.options(header='True', delimiter=',') \
  .csv("/tmp/resources/zipcodes.csv")
df3.printSchema()

schema = StructType() \
    .add("RecordNumber",IntegerType(),True) \
    .add("Zipcode",IntegerType(),True) \
    .add("ZipCodeType",StringType(),True) \
    .add("City",StringType(),True) \
    .add("State",StringType(),True) \
    .add("LocationType",StringType(),True) \
    .add("Lat",DoubleType(),True) \
    .add("Long",DoubleType(),True) \
    .add("Xaxis",IntegerType(),True) \
    .add("Yaxis",DoubleType(),True) \
    .add("Zaxis",DoubleType(),True) \
    .add("WorldRegion",StringType(),True) \
    .add("Country",StringType(),True) \
    .add("LocationText",StringType(),True) \
    .add("Location",StringType(),True) \
    .add("Decommisioned",BooleanType(),True) \
    .add("TaxReturnsFiled",StringType(),True) \
    .add("EstimatedPopulation",IntegerType(),True) \
    .add("TotalWages",IntegerType(),True) \
    .add("Notes",StringType(),True)

df_with_schema = spark.read.format("csv") \
    .option("header", True) \
    .schema(schema) \
    .load(/tmp/resources/zipcodes.csv")
df_with_schema.printSchema()

df2.write.option("header",True) \
 .csv("/tmp/spark_output/zipcodes123")
 ================================================================================
 Complete Example of PySpark read and write Parquet file

import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName("parquetFile").getOrCreate()
data =[("James ","","Smith","36636","M",3000),
          ("Michael ","Rose","","40288","M",4000),
          ("Robert ","","Williams","42114","M",4000),
          ("Maria ","Anne","Jones","39192","F",4000),
          ("Jen","Mary","Brown","","F",-1)]
columns=["firstname","middlename","lastname","dob","gender","salary"]
```

```
df=spark.createDataFrame(data,columns)
df.write.mode("overwrite").parquet("/tmp/output/people.parquet")
parDF1=spark.read.parquet("/tmp/output/people.parquet")
parDF1.createOrReplaceTempView("parquetTable")
parDF1.printSchema()
parDF1.show(truncate=False)

parkSQL = spark.sql("select * from ParquetTable where salary >= 4000 ")
parkSQL.show(truncate=False)

spark.sql("CREATE TEMPORARY VIEW PERSON USING parquet OPTIONS (path \"/tmp/output/people.parquet\")")
spark.sql("SELECT * FROM PERSON").show()

df.write.partitionBy("gender","salary").mode("overwrite").parquet("/tmp/output/people2.parquet")

parDF2=spark.read.parquet("/tmp/output/people2.parquet/gender=M")
parDF2.show(truncate=False)

spark.sql("CREATE TEMPORARY VIEW PERSON2 USING parquet OPTIONS (path \"/tmp/output/people2.parquet/gender=F\")")
spark.sql("SELECT * FROM PERSON2" ).show()
```

==========

Real time programming examples:

---

convert-column-python-list.py:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[1]") \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

data = [("James","Smith","USA","CA"),("Michael","Rose","USA","NY"), \
   ("Robert","Williams","USA","CA"),("Maria","Jones","USA","FL") \
 ]
columns=["firstname","lastname","country","state"]
df=spark.createDataFrame(data=data,schema=columns)
df.show()
print(df.collect())

states1=df.rdd.map(lambda x: x[3]).collect()
print(states1)
#['CA', 'NY', 'CA', 'FL']
from collections import OrderedDict
res = list(OrderedDict.fromkeys(states1))
print(res)
#['CA', 'NY', 'FL']


#Example 2
states2=df.rdd.map(lambda x: x.state).collect()
print(states2)
#['CA', 'NY', 'CA', 'FL']

states3=df.select(df.state).collect()
print(states3)
#[Row(state='CA'), Row(state='NY'), Row(state='CA'), Row(state='FL')]

states4=df.select(df.state).rdd.flatMap(lambda x: x).collect()
print(states4)
#['CA', 'NY', 'CA', 'FL']

states5=df.select(df.state).toPandas()['state']
states6=list(states5)
print(states6)
#['CA', 'NY', 'CA', 'FL']

pandDF=df.select(df.state,df.firstname).toPandas()
print(list(pandDF['state']))
print(list(pandDF['firstname']))
```

CurreentDate.py:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.functions import to_timestamp, current_timestamp
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, LongType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
```

```
schema = StructType([
        StructField("seq", StringType(), True)])

dates = ['1']

df = spark.createDataFrame(list('1'), schema=schema)

df.show()
```

Pandas pyspark dataframe:

```
import pandas as pd
data = [['Scott', 50], ['Jeff', 45], ['Thomas', 54],['Ann',34]]

# Create the pandas DataFrame
pandasDF = pd.DataFrame(data, columns = ['Name', 'Age'])

# print dataframe.
print(pandasDF)

from pyspark.sql import SparkSession

spark = SparkSession.builder \
   .master("local[1]") \
   .appName("SparkByExamples.com") \
   .getOrCreate()

sparkDF=spark.createDataFrame(pandasDF)
sparkDF.printSchema()
sparkDF.show()

#sparkDF=spark.createDataFrame(pandasDF.astype(str))
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
mySchema = StructType([ StructField("First Name", StringType(), True)\
                  ,StructField("Age", IntegerType(), True)])

sparkDF2 = spark.createDataFrame(pandasDF,schema=mySchema)
sparkDF2.printSchema()
sparkDF2.show()


spark.conf.set("spark.sql.execution.arrow.enabled","true")
spark.conf.set("spark.sql.execution.arrow.pyspark.fallback.enabled","true")

pandasDF2=sparkDF2.select("*").toPandas
print(pandasDF2)


test=spark.conf.get("spark.sql.execution.arrow.enabled")
print(test)

test123=spark.conf.get("spark.sql.execution.arrow.pyspark.fallback.enabled")
print(test123)
```
---
pyspark add_month date:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

from pyspark.sql.functions import col,expr
data=[("2019-01-23",1),("2019-06-24",2),("2019-09-20",3)]
spark.createDataFrame(data).toDF("date","increment") \
   .select(col("date"),col("increment"), \
     expr("add_months(to_date(date,'yyyy-MM-dd'),cast(increment as int))").alias("inc_date")) \
   .show()
```
---
pyspark add new column:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
                .appName('SparkByExamples.com') \
                .getOrCreate()

data = [('James','Smith','M',3000),
  ('Anna','Rose','F',4100),
  ('Robert','Williams','M',6200),
```

```
]

columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()


if 'salary1' not in df.columns:
   print("aa")

# Add new constanct column
from pyspark.sql.functions import lit
df.withColumn("bonus_percent", lit(0.3)) \
  .show()

#Add column from existing column
df.withColumn("bonus_amount", df.salary*0.3) \
  .show()

#Add column by concatinating existing columns
from pyspark.sql.functions import concat_ws
df.withColumn("name", concat_ws(",","firstname",'lastname')) \
  .show()

#Add current date
from pyspark.sql.functions import current_date
df.withColumn("current_date", current_date()) \
  .show()


from pyspark.sql.functions import when
df.withColumn("grade", \
   when((df.salary < 4000), lit("A")) \
     .when((df.salary >= 4000) & (df.salary <= 5000), lit("B")) \
     .otherwise(lit("C")) \
  ).show()

# Add column using select
df.select("firstname","salary", lit(0.3).alias("bonus")).show()
df.select("firstname","salary", lit(df.salary * 0.3).alias("bonus_amount")).show()
df.select("firstname","salary", current_date().alias("today_date")).show()

#Add columns using SQL
df.createOrReplaceTempView("PER")
spark.sql("select firstname,salary, '0.3' as bonus from PER").show()
spark.sql("select firstname,salary, salary * 0.3 as bonus_amount from PER").show()
spark.sql("select firstname,salary, current_date() as today_date from PER").show()
spark.sql("select firstname,salary, " +
        "case salary when salary < 4000 then 'A' "+
        "else 'B' END as grade from PER").show()
-------------------------------------------------------------------------------------------
pyspark-aggregate.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import approx_count_distinct,collect_list
from pyspark.sql.functions import collect_set,sum,avg,max,countDistinct,count
from pyspark.sql.functions import first, last, kurtosis, min, mean, skewness
from pyspark.sql.functions import stddev, stddev_samp, stddev_pop, sumDistinct
from pyspark.sql.functions import variance,var_samp,  var_pop

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James", "Sales", 3000),
   ("Michael", "Sales", 4600),
   ("Robert", "Sales", 4100),
   ("Maria", "Finance", 3000),
   ("James", "Sales", 3000),
   ("Scott", "Finance", 3300),
   ("Jen", "Finance", 3900),
   ("Jeff", "Marketing", 3000),
   ("Kumar", "Marketing", 2000),
   ("Saif", "Sales", 4100)
 ]
schema = ["employee_name", "department", "salary"]


df = spark.createDataFrame(data=simpleData, schema = schema)
```

```
df.printSchema()
df.show(truncate=False)

print("approx_count_distinct: " + \
     str(df.select(approx_count_distinct("salary")).collect()[0][0]))

print("avg: " + str(df.select(avg("salary")).collect()[0][0]))

df.select(collect_list("salary")).show(truncate=False)

df.select(collect_set("salary")).show(truncate=False)

df2 = df.select(countDistinct("department", "salary"))
df2.show(truncate=False)
print("Distinct Count of Department &amp;amp; Salary: "+str(df2.collect()[0][0]))

print("count: "+str(df.select(count("salary")).collect()[0]))
df.select(first("salary")).show(truncate=False)
df.select(last("salary")).show(truncate=False)
df.select(kurtosis("salary")).show(truncate=False)
df.select(max("salary")).show(truncate=False)
df.select(min("salary")).show(truncate=False)
df.select(mean("salary")).show(truncate=False)
df.select(skewness("salary")).show(truncate=False)
df.select(stddev("salary"), stddev_samp("salary"), \
   stddev_pop("salary")).show(truncate=False)
df.select(sum("salary")).show(truncate=False)
df.select(sumDistinct("salary")).show(truncate=False)
df.select(variance("salary"),var_samp("salary"),var_pop("salary")) \
  .show(truncate=False)
--------------------------------------------------------------------------------
pyspark-array_string:

import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[1]") \
                .appName('SparkByExamples.com') \
                .getOrCreate()

columns = ["name","languagesAtSchool","currentState"]
data = [("James,,Smith",["Java","Scala","C++"],"CA"), \
   ("Michael,Rose,",["Spark","Java","C++"],"NJ"), \
   ("Robert,,Williams",["CSharp","VB"],"NV")]

df = spark.createDataFrame(data=data,schema=columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import col, concat_ws
df2 = df.withColumn("languagesAtSchool",
  concat_ws(",",col("languagesAtSchool")))
df2.printSchema()
df2.show(truncate=False)


df.createOrReplaceTempView("ARRAY_STRING")
spark.sql("select name, concat_ws(',',languagesAtSchool) as languagesAtSchool,currentState from ARRAY_STRING")
.show(truncate=False)
--------------------------------------------------------------------------------------------------------------
pyspark type.py:

from pyspark.sql import SparkSession
from pyspark.sql.types import StringType, ArrayType,StructType,StructField
spark = SparkSession.builder \
                .appName('SparkByExamples.com') \
                .getOrCreate()


arrayCol = ArrayType(StringType(),False)

data = [
 ("James,,Smith",["Java","Scala","C++"],["Spark","Java"],"OH","CA"),
 ("Michael,Rose,",["Spark","Java","C++"],["Spark","Java"],"NY","NJ"),
 ("Robert,,Williams",["CSharp","VB"],["Spark","Python"],"UT","NV")
]

schema = StructType([
   StructField("name",StringType(),True),
```

```
    StructField("languagesAtSchool",ArrayType(StringType()),True),
    StructField("languagesAtWork",ArrayType(StringType()),True),
    StructField("currentState", StringType(), True),
    StructField("previousState", StringType(), True)
  ])

df = spark.createDataFrame(data=data,schema=schema)
df.printSchema()
df.show()

from pyspark.sql.functions import explode
df.select(df.name,explode(df.languagesAtSchool)).show()

from pyspark.sql.functions import split
df.select(split(df.name,",").alias("nameAsArray")).show()

from pyspark.sql.functions import array
df.select(df.name,array(df.currentState,df.previousState).alias("States")).show()

from pyspark.sql.functions import array_contains
df.select(df.name,array_contains(df.languagesAtSchool,"Java")
    .alias("array_contains")).show()
```

Broad cast dataframe:

```
import pyspark
from pyspark.sql import SparkSession


spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

states = {"NY":"New York", "CA":"California", "FL":"Florida"}
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),
    ("Michael","Rose","USA","NY"),
    ("Robert","Williams","USA","CA"),
    ("Maria","Jones","USA","FL")
  ]

columns = ["firstname","lastname","country","state"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

def state_convert(code):
    return broadcastStates.value[code]

result = df.rdd.map(lambda x: (x[0],x[1],x[2],state_convert(x[3]))).toDF(columns)
result.show(truncate=False)

# Broadcast variable on filter

filteDf= df.where((df['state'].isin(broadcastStates.value)))
```

---

pyspark cast columns.py:

```
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James",34,"2006-01-01","true","M",3000.60),
    ("Michael",33,"1980-01-10","true","F",3300.80),
    ("Robert",37,"06-01-1992","false","M",5000.50)
  ]

columns = ["firstname","age","jobStartDate","isGraduated","gender","salary"]
df = spark.createDataFrame(data = simpleData, schema = columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import col
from pyspark.sql.types import StringType,BooleanType,DateType
df2 = df.withColumn("age",col("age").cast(StringType())) \
    .withColumn("isGraduated",col("isGraduated").cast(BooleanType())) \
    .withColumn("jobStartDate",col("jobStartDate").cast(DateType()))
df2.printSchema()
```

```
df3 = df2.selectExpr("cast(age as int) age",
   "cast(isGraduated as string) isGraduated",
   "cast(jobStartDate as string) jobStartDate")
df3.printSchema()
df3.show(truncate=False)

df3.createOrReplaceTempView("CastExample")
df4 = spark.sql("SELECT STRING(age),BOOLEAN(isGraduated),DATE(jobStartDate) from CastExample")
df4.printSchema()
df4.show(truncate=False)
```

---

pyspark chanfe-doubletype.py:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import DoubleType, IntegerType
# Create SparkSession
spark = SparkSession.builder \
        .appName('SparkByExamples.com') \
        .getOrCreate()

simpleData = [("James","34","true","M","3000.6089"),
   ("Michael","33","true","F","3300.8067"),
   ("Robert","37","false","M","5000.5034")
 ]

columns = ["firstname","age","isGraduated","gender","salary"]
df = spark.createDataFrame(data = simpleData, schema = columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import col,round,expr
df.withColumn("salary",df.salary.cast('double')).printSchema()
df.withColumn("salary",df.salary.cast(DoublerType())).printSchema()
df.withColumn("salary",col("salary").cast('double')).printSchema()

#df.withColumn("salary",round(df.salary.cast(DoubleType()),2)).show(truncate=False).printSchema()
df.selectExpr("firstname","isGraduated","cast(salary as double) salary").printSchema()

df.createOrReplaceTempView("CastExample")
spark.sql("SELECT firstname,isGraduated,DOUBLE(salary) as salary from CastExample").printSchema()


#df.select("firstname",expr(df.age),"isGraduated",col("salary").cast('float').alias("salary")).show()
```

---

collect.py:

```
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

dept = [("Finance",10), \
   ("Marketing",20), \
   ("Sales",30), \
   ("IT",40) \
 ]
deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF.printSchema()
deptDF.show(truncate=False)

dataCollect = deptDF.collect()

print(dataCollect)

dataCollect2 = deptDF.select("dept_name").collect()
print(dataCollect2)

for row in dataCollect:
   print(row['dept_name'] + "," +str(row['dept_id']))
```

---

column function.py:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data=[("James","Bond","100",None),
```

```
    ("Ann","Varsa","200",'F'),
    ("Tom Cruise","XXX","400",''),
    ("Tom Brand",None,"400",'M')]
columns=["fname","lname","id","gender"]
df=spark.createDataFrame(data,columns)

#alias
from pyspark.sql.functions import expr
df.select(df.fname.alias("first_name"), \
        df.lname.alias("last_name"), \
        expr(" fname ||','|| lname").alias("fullName") \
  ).show()

#asc, desc
df.sort(df.fname.asc()).show()
df.sort(df.fname.desc()).show()

#cast
df.select(df.fname,df.id.cast("int")).printSchema()

#between
df.filter(df.id.between(100,300)).show()

#contains
df.filter(df.fname.contains("Cruise")).show()

#startswith, endswith()
df.filter(df.fname.startswith("T")).show()
df.filter(df.fname.endswith("Cruise")).show()

#eqNullSafe

#isNull & isNotNull
df.filter(df.lname.isNull()).show()
df.filter(df.lname.isNotNull()).show()

#like , rlike
df.select(df.fname,df.lname,df.id) \
  .filter(df.fname.like("%om"))

#over

#substr
df.select(df.fname.substr(1,2).alias("substr")).show()

#when & otherwise
from pyspark.sql.functions import when
df.select(df.fname,df.lname,when(df.gender=="M","Male") \
           .when(df.gender=="F","Female") \
           .when(df.gender==None ,"") \
           .otherwise(df.gender).alias("new_gender") \
   ).show()

#isin
li=["100","200"]
df.select(df.fname,df.lname,df.id) \
  .filter(df.id.isin(li)) \
  .show()

from pyspark.sql.types import StructType,StructField,StringType,ArrayType,MapType
data=[(("James","Bond"),["Java","C#"],{'hair':'black','eye':'brown'}),
    (("Ann","Varsa"),[".NET","Python"],{'hair':'brown','eye':'black'}),
    (("Tom Cruise",""),["Python","Scala"],{'hair':'red','eye':'grey'}),
    (("Tom Brand",None),["Perl","Ruby"],{'hair':'black','eye':'blue'})]

schema = StructType([
      StructField('name', StructType([
         StructField('fname', StringType(), True),
         StructField('lname', StringType(), True)])),
      StructField('languages', ArrayType(StringType()),True),
      StructField('properties', MapType(StringType(),StringType()),True)
   ])
df=spark.createDataFrame(data,schema)
df.printSchema()
#getItem()
df.select(df.languages.getItem(1)).show()

df.select(df.properties.getItem("hair")).show()
```

```
#getField from Struct or Map
df.select(df.properties.getField("hair")).show()

df.select(df.name.getField("fname")).show()

#dropFields
#from pyspark.sql.functions import col
#df.withColumn("name1",col("name").dropFields(["fname"])).show()

#withField
#from pyspark.sql.functions import lit
#df.withColumn("name",df.name.withField("fname",lit("AA"))).show()

#from pyspark.sql import Row
#from pyspark.sql.functions import lit
#df = spark.createDataFrame([Row(a=Row(b=1, c=2))])
#df.withColumn('a', df['a'].withField('b', lit(3))).select('a.b').show()

#from pyspark.sql import Row
#from pyspark.sql.functions import col, lit
#df = spark.createDataFrame([
#Row(a=Row(b=1, c=2, d=3, e=Row(f=4, g=5, h=6)))])
#df.withColumn('a', df['a'].dropFields('b')).show()
```

---

column operations.py:

```
from pyspark.sql import SparkSession,Row
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data=[("James",23),("Ann",40)]
df=spark.createDataFrame(data).toDF("name.fname","gender")
df.printSchema()
df.show()

from pyspark.sql.functions import col
df.select(col("`name.fname`")).show()
df.select(df["`name.fname`"]).show()
df.withColumn("new_col",col("`name.fname`").substr(1,2)).show()
df.filter(col("`name.fname`").startswith("J")).show()
new_cols=(column.replace('.', '_') for column in df.columns)
df2 = df.toDF(*new_cols)
df2.show()


# Using DataFrame object
df.select(df.gender).show()
df.select(df["gender"]).show()
#Accessing column name with dot (with backticks)
df.select(df["`name.fname`"]).show()

#Using SQL col() function
from pyspark.sql.functions import col
df.select(col("gender")).show()
#Accessing column name with dot (with backticks)
df.select(col("`name.fname`")).show()

#Access struct column
data=[Row(name="James",prop=Row(hair="black",eye="blue")),
    Row(name="Ann",prop=Row(hair="grey",eye="black"))]
df=spark.createDataFrame(data)
df.printSchema()

df.select(df.prop.hair).show()
df.select(df["prop.hair"]).show()
df.select(col("prop.hair")).show()
df.select(col("prop.*")).show()

# Column operators
data=[(100,2,1),(200,3,4),(300,4,4)]
df=spark.createDataFrame(data).toDF("col1","col2","col3")
df.select(df.col1 + df.col2).show()
df.select(df.col1 - df.col2).show()
df.select(df.col1 * df.col2).show()
df.select(df.col1 / df.col2).show()
df.select(df.col1 % df.col2).show()

df.select(df.col2 > df.col3).show()
```

```
df.select(df.col2 < df.col3).show()
df.select(df.col2 == df.col3).show()
```

---

convert map to columns.py:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

dataDictionary = [
      ('James',{'hair':'black','eye':'brown'}),
      ('Michael',{'hair':'brown','eye':None}),
      ('Robert',{'hair':'red','eye':'black'}),
      ('Washington',{'hair':'grey','eye':'grey'}),
      ('Jefferson',{'hair':'brown','eye':''})
      ]

df = spark.createDataFrame(data=dataDictionary, schema = ['name','properties'])
df.printSchema()
df.show(truncate=False)

df3=df.rdd.map(lambda x: \
   (x.name,x.properties["hair"],x.properties["eye"])) \
   .toDF(["name","hair","eye"])
df3.printSchema()
df3.show()

df.withColumn("hair",df.properties.getItem("hair")) \
  .withColumn("eye",df.properties.getItem("eye")) \
  .drop("properties") \
  .show()

df.withColumn("hair",df.properties["hair"]) \
  .withColumn("eye",df.properties["eye"]) \
  .drop("properties") \
  .show()

# Functions
from pyspark.sql.functions import explode,map_keys,col
keysDF = df.select(explode(map_keys(df.properties))).distinct()
keysList = keysDF.rdd.map(lambda x:x[0]).collect()
keyCols = list(map(lambda x: col("properties").getItem(x).alias(str(x)), keysList))
df.select(df.name, *keyCols).show()
```

---

columns to map.py:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data = [ ("36636","Finance",3000,"USA"),
   ("40288","Finance",5000,"IND"),
   ("42114","Sales",3900,"USA"),
   ("39192","Marketing",2500,"CAN"),
   ("34534","Sales",6500,"USA") ]
schema = StructType([
    StructField('id', StringType(), True),
    StructField('dept', StringType(), True),
    StructField('salary', IntegerType(), True),
    StructField('location', StringType(), True)
    ])

df = spark.createDataFrame(data=data,schema=schema)
df.printSchema()
df.show(truncate=False)

#Convert scolumns to Map
from pyspark.sql.functions import col,lit,create_map
df = df.withColumn("propertiesMap",create_map(
      lit("salary"),col("salary"),
      lit("location"),col("location")
      )).drop("salary","location")
df.printSchema()
df.show(truncate=False)
```

Real Time Projects:

permonth and per year query:

```
SELECT date_format(timestamp_column, 'yyyy-MM') as month_year, count(*) as count
FROM table_name
GROUP BY month_year
```

From pyspark.sql.types import *

The pyspark.sql.types module in PySpark provides a collection of data types that can be used to define the schema of a DataFrame. Here's a brief explanation of some of the data types available in this module:

StringType: Represents string values
IntegerType: Represents integer values
LongType: Represents long integer values
DoubleType: Represents double precision floating-point values
FloatType: Represents single precision floating-point values
DecimalType: Represents decimal values with fixed precision and scale
TimestampType: Represents timestamp values
DateType: Represents date values
BooleanType: Represents boolean values
You can use these data types to define the schema of a DataFrame using the StructType and StructField classes.

For example:

ETL=>Ingestion transfermatuon load reporting:

you can create a DataFrame using the createDataFrame method.

```
cust_data='dbfs:/filestor/data/cust.csv',
emp_data='dbfs:/filestore/data/emp.csv',
dept_data='dbfs:/filestore/data/dept.csv'

=>cust_schema=StructType([StructField('cust_no',IntegerType(),Nullabale=true)])

schema = StructType([
   StructField("name", StringType(), True),
   StructField("age", IntegerType(), True),
   StructField("salary", DoubleType(), True)
])
```

RDD:

```
from pysark.sql.tyoes import *
from pyspark.sql import sparksession

schema = StructType([
   StructField("name", StringType(), True),
   StructField("age", IntegerType(), True),
   StructField("salary", DoubleType(), True)
])


from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Example").getOrCreate()

data = [("Alice", 25, 50000.0), ("Bob", 30, 60000.0), ("Charlie", 35, 70000.0)]
rdd = spark.sparkContext.parallelize(data)

df = spark.createDataFrame(rdd, schema)
df.show()
```

```
 from pyspark.sql.functions import *
```
statement imports all the functions available in the pyspark.sql.functions module in PySpark.
This allows you to use these functions in your PySpark SQL queries without having to prefix them with f..

Here are some common functions that can be used with this import statement:

col: returns a Column based on the given column name
lit: creates a Column with a literal value
when: evaluates a condition and returns one of two values based on the result
concat: concatenates two or more columns into a single column
substring: extracts a substring from a column based on a starting index and length
upper: converts a string column to uppercase
lower: converts a string column to lowercase
trim: removes leading and trailing whitespace from a string column
round: rounds a numeric column to a specified number of decimal places

count: counts the number of rows in a DataFrame or the number of non-null values in a specific column
sum: computes the sum of values in a numeric column
avg: computes the average of values in a numeric column
min: finds the minimum value in a numeric or string column
max: finds the maximum value in a numeric or string column

By importing all the functions using from pyspark.sql.functions import *, you can use these and many other functions in your PySpark SQL queries. However, it's important to note that importing all functions can lead to naming conflicts if you have functions with the same name as those in the module. In such cases, it may be better to import functions individually to avoid any potential issues.

from pyspark.sql.functions import *
each examples
Sure, here are some examples of how to use some of the functions available in pyspark.sql.functions:

col: Returns a Column based on the given column name.
sql
from pyspark.sql.functions import col

```
# Create a DataFrame with a column named "name"
data = [("Alice", 25), ("Bob", 30), ("Charlie", 35)]
df = spark.createDataFrame(data, ["name", "age"])

# Use "col" to reference the "name" column in a PySpark SQL query
df.select(col("name")).show()
```

---
when: Evaluates a condition and returns one of two values based on the result.

```
from pyspark.sql.functions import when

# Create a DataFrame with columns named "age" and "is_adult"

data = [(25, True), (17, False), (30, True)]
df = spark.createDataFrame(data, ["age", "is_adult"])

# Use "when" to create a new column named "adult_status"
# If "is_adult" is True, set "adult_status" to "Adult"
# If "is_adult" is False, set "adult_status" to "Minor"

df.select(
   "age",
   "is_adult",
   when(col("is_adult"), "Adult").otherwise("Minor").alias("adult_status")
).show()
```

---

concat: Concatenates two or more columns into a single column.
sql

```
from pyspark.sql.functions import concat

# Create a DataFrame with columns named "first_name" and "last_name"

data = [("Alice", "Smith"), ("Bob", "Johnson"), ("Charlie", "Brown")]
df = spark.createDataFrame(data, ["first_name", "last_name"])

# Use "concat" to create a new column named "full_name"
# Concatenate the "first_name" and "last_name" columns with a space in between

df.select(concat(col("first_name"), lit(" "), col("last_name")).alias("full_name")).show()
```

---
substring: Extracts a substring from a column based on a starting index and length.
sql

```
from pyspark.sql.functions import substring

# Create a DataFrame with a column named "name"

data = [("Alice Smith"), ("Bob Johnson"), ("Charlie Brown")]
df = spark.createDataFrame(data, ["name"])

# Use "substring" to create a new column named "last_name"
# Extract the last name from the "name" column, assuming it is separated by a space

df.select(substring(col("name"), instr(col("name"), " ") + 1, length(col("name"))).alias("last_name")).show()
```

ETL-CONCEPT:

#ETL--concepts etc

```
from pyspark.sql.types import *
from pyspark.sql.functions import col

data_shopping_cust = 'dbfs:/FileStore/Shopping_CustomerData.csv'
data_shopping_index = 'dbfs:/FileStore/Shopping_ShoppingIndexData.csv'

cust_schema = StructType([StructField('CustomerAge', IntegerType(), nullable=True)])
index_schema = StructType([StructField('Kolkata', IntegerType(), nullable=True)])

df = spark.read.option('header', 'true').csv(path=data_shopping_cust, schema=cust_schema).show()
df1 = spark.read.option('header', 'true').csv(path=data_shopping_index, schema=index_schema).show()
```

---

partitions:

```
from pyspark.sql.functions import year

df = spark.read.format('csv').option('header', 'true').load('dbfs:/FileStore/Shopping_CustomerData.csv')
df = df.withColumn('year', year('AnnualIncome',))
df.write.partitionBy('year').mode('overwrite').parquet('/dbfs:/FileStore/output.csv')
```

Using hash:

```
df_partion = spark.read.format('csv').option('header', 'true').load('dbfs:/FileStore/Shopping_CustomerData.csv')
df_partion  = df.repartition(4)
df_partion .write.mode('overwrite').parquet('File//path/to/output')
df_partion .show()
```

---

partition:

```
from pyspark.sql.types import *

# Write the DataFrame to disk, partitioned by "Channel_Name" and "Genre"
df.write.mode("overwrite").partitionBy("Channel_Name", "Genre").csv("dbfs:/FileStore/output")

# Read the data back from disk
df2 = spark.read.csv("dbfs:/FileStore/output", header=True)

# Show the data
df2.show()
```

```
# Print the current number of partitions
print('Number of current partitions:', str(df.rdd.getNumPartitions()))

# Reduce the number of partitions to 5 using coalesce
df_reduced = df.coalesce(5)
print('Number of partitions after reducing using coalesce:', str(df_reduced.rdd.getNumPartitions()))

# Increase the number of partitions to 10 using repartition
df_increased = df.repartition(10)
print('Number of partitions after increasing using repartition:', str(df_increased.rdd.getNumPartitions()))
```

Broad Cast Join:

```
from pyspark.sql.functions import broadcast

large_df = spark.read.parquet("path/to/large/dataframe")
small_df = spark.read.parquet("path/to/small/dataframe")

# Mark the small_df for broadcast join
small_df_b = broadcast(small_df)

# Perform the join operation
joined_df = large_df.join(small_df_b, "join_column")
```

```
from pyspark.sql.functions import broadcast

df1 = spark.read.format("csv").load("path/to/df1.csv")
df2 = spark.read.format("csv").load("path/to/df2.csv")

joined_df = df1.join(broadcast(df2), "join_key")
```

```
from pyspark.sql.functions import broadcast
from pyspark.sql.types import StructType, StructField, IntegerType

df = spark.read.format("csv").load("path/to/df.csv")
```

```
broadcast_var = sc.broadcast([1, 2, 3])

joined_df = df.filter(df.col("join_key").isin(broadcast_var.value)) \
            .join(broadcast(df), "join_key")
```

SAS code converted into pyspark:

```
# SAS code

DATA mydata;
  INFILE 'path/to/file.csv' delimiter=',' DSD;
  INPUT var1 var2 var3;
RUN;

#pyspark code

from pyspark.sql.functions import *

mydata = spark.read.format("csv") \
   .option("header", "false") \
   .option("delimiter", ",") \
   .option("inferSchema", "true") \
   .load("path/to/file.csv") \
   .toDF("var1", "var2", "var3")
```

Merge a two data sets:

```
from pyspark.sql.functions import *

dataset1 = spark.read.format("csv").load("path/to/dataset1.csv").toDF("var1", "var2")
dataset2 = spark.read.format("csv").load("path/to/dataset2.csv").toDF("var1", "var3")

merged = dataset1.join(dataset2, "var1", "inner") \
             .select("var1", "var2", "var3")
```

```
.option('mode','permisive'),option('mode','dropmalformed') and option('mode','failfast')

permisive mode:to parse the row and columns without malformed and its continues
dropmalformed:to drop the mal formd row and columns values
failfast:stop the execution process if any error occures showing the values

df = spark.read.format("csv") \
        .option("header", "true") \
        .option("mode", "permissive") \
        .load("path/to/file.csv")
```

Build a ONE data warehouse in pyspark:

```
from pyspark.sql import SparkSession

# create a SparkSession
spark = SparkSession.builder.appName("Data Warehouse").getOrCreate()

# define the schema for the data warehouse tables
schema = "id INT, name STRING, age INT, gender STRING, city STRING"

# load data from a CSV file into a PySpark dataframe
df = spark.read.format("csv").option("header", True).load("path/to/csv/file.csv")

# apply transformations to the data
df_filtered = df.filter(df.age > 18)
df_grouped = df_filtered.groupBy(df_filtered.city).agg({"age": "avg"})

# write data to the data warehouse table

df_grouped.write.format("jdbc")
.option("url", "jdbc:postgresql://localhost:5432/mydatabase").option("dbtable", "mytable")
.option("user", "myuser").option("password", "mypassword").mode("append").save()
```

Spark Sql In Data Bricks:

```
# Create a DataFrame from a CSV file
data = spark.read.format("csv").option("header", "true").load("/FileStore/tables/data.csv")

# Register the DataFrame as a temporary table
data.createOrReplaceTempView("my_table")
```

```
# Write a SQL query to select data from the table
result = spark.sql("SELECT * FROM my_table WHERE column1 = 'value'")

#Joining twon data frame using sql

join_data=spark.sql'selectr* from table1 join  table2 on table1.tableId==table2.table2Id')

# Display the results of the query
display(result)

Q:
--
df=spark.sql("select e.ename,d.dname from emp  as e join dept as d on e.eno==d.dno").show()
-----------------------------------------------------------------------------------------------------------------------------------

df = spark.sql("SELECT ename, SUM(esal) AS totalsal FROM emp GROUP BY ename")
df.show()


df=spark.sql("select count(*),sum(esal) from  emp where esal>=18000").show()


df=spark.sql("select distinct e.esal, e.ename,d.dname from emp as e left join dept as d on e.eno==d.dno").show()
-----------------------------------------------------------------------------------------------------------------------------------

SELECT A.TABLE_SCHEMA, B.TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.TABLES AS A
INNER JOIN INFORMATION_SCHEMA.COLUMNS AS B
ON A.TABLE_NAME = B.TABLE_NAME AND A.TABLE_SCHEMA = B.TABLE_SCHEMA
WHERE COLUMN_NAME LIKE '%FirstName%' AND TABLE_TYPE = 'BASE TABLE';

-----------------------------------------------------------------------------------------
How to get the list of tables in a common columnname:
-----------------------------------------------------
SELECT TABLE_NAME, COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE COLUMN_NAME = 'FirstName'
GROUP BY TABLE_NAME, COLUMN_NAME
HAVING COUNT(*) > 1  note:i have taken for all varchar data type etc
-------------------------------------------------------------------------------

three tables joins query:
-------------------------
SELECT *FROM Person.Person AS p INNER JOIN HumanResources.Employee AS e ON p.ModifiedDate = e.ModifiedDate
INNER JOIN Production.ProductReview AS r
 ON e.BusinessEntityID = r.ProductReviewID
 -----------------------------------------------------------------------------------------------------------------------------

SELECT A.TABLE_SCHEMA, B.TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.TABLES AS A
INNER JOIN INFORMATION_SCHEMA.COLUMNS AS B
   ON A.TABLE_NAME = B.TABLE_NAME AND A.TABLE_SCHEMA = B.TABLE_SCHEMA
WHERE COLUMN_NAME LIKE '%EmailAddress%' AND TABLE_TYPE = 'BASE TABLE';

SELECT TABLE_NAME, COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE COLUMN_NAME LIKE '%rowguid%'



SELECT TABLE_NAME, COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE COLUMN_NAME = 'BusinessEntityID'
GROUP BY TABLE_NAME, COLUMN_NAME
HAVING COUNT(*) > =1


SELECT *FROM Person.Address AS p
INNER JOIN HumanResources.Employee AS e ON p.ModifiedDate = e.ModifiedDate
INNER JOIN Production.ProductReview AS r ON e.BusinessEntityID = r.ProductReviewID
inner join Person.EmailAddress as ea on ea.BusinessEntityID=e.BusinessEntityID
inner join Person.PersonPhone as pp on ea.BusinessEntityID=pp.BusinessEntityID
inner join Person.PhoneNumberType paa on paa.PhoneNumberTypeID=pp.PhoneNumberTypeID
inner join Person.Person as pt on pt.BusinessEntityID=pp.BusinessEntityID
-----------------------------------------------------------------------------------------------------------------------------------

SELECT *FROM Person.Address AS p
```

```
INNER JOIN HumanResources.Employee AS e ON p.ModifiedDate = e.ModifiedDate
INNER JOIN Production.ProductReview AS r ON e.BusinessEntityID = r.ProductReviewID
inner join Person.EmailAddress as ea on ea.BusinessEntityID=e.BusinessEntityID
inner join Person.PersonPhone as pp on ea.BusinessEntityID=pp.BusinessEntityID
inner join Person.PhoneNumberType paa on paa.PhoneNumberTypeID=pp.PhoneNumberTypeID
inner join Person.Person as pt on pt.BusinessEntityID=pp.BusinessEntityID

select
Title,
FirstName,
MiddleName,
LastName,
NameStyle,
BirthDate,
MaritalStatus,
Suffix,
Gender,
AddressLine1,
AddressLine2

FROM Person.Address AS p
INNER JOIN HumanResources.Employee AS e ON p.ModifiedDate = e.ModifiedDate
INNER JOIN Production.ProductReview AS r ON e.BusinessEntityID = r.ProductReviewID
inner join Person.EmailAddress as ea on ea.BusinessEntityID=e.BusinessEntityID
inner join Person.PersonPhone as pp on ea.BusinessEntityID=pp.BusinessEntityID
inner join Person.PhoneNumberType paa on paa.PhoneNumberTypeID=pp.PhoneNumberTypeID
inner join Person.Person as pt on pt.BusinessEntityID=pp.BusinessEntityID
-------------------------------------------------------------------------------------
select * from person.StateProvince
select * from person.Address
select * from person.CountryRegion
select * from Sales.CountryRegionCurrency
select * from Sales.SalesTerritory

SELECT Person.StateProvince.StateProvinceCode, Person.StateProvince.CountryRegionCode,
Person.Address.City,Person.Address.PostalCode,Person.Address.AddressID
FROM Person.StateProvince
INNER JOIN Person.Address ON Person.StateProvince.StateProvinceID=Person.Address.StateProvinceID
FULL OUTER JOIN Person.CountryRegion ON Person.Address.ModifiedDate = Person.CountryRegion.ModifiedDate
INNER JOIN Sales.CountryRegionCurrency ON Person.StateProvince.CountryRegionCode = Sales.CountryRegionCurrency.CountryRegionCode
INNER JOIN Sales.SalesTerritory ON Person.StateProvince.CountryRegionCode = Sales.SalesTerritory.CountryRegionCode
-------------------------------------------------------------------------------------

OutPut:
--------

NULL  Ken   J     Sánchez        0     1969-01-29  S     NULL  M     9505 Hargate Court     NULL
NULL  Terri Lee   Duffy  0      1971-08-01  S     NULL  F     9505 Hargate Court     NULL
NULL  Roberto     NULL  Tamburello  0     1974-11-12  M     NULL  M     9505 Hargate Court     NULL
NULL  Rob   NULL  Walters 0     1974-12-23  S     NULL  M     9505 Hargate Court     NULL
NULL  Ken   J     Sánchez        0     1969-01-29  S     NULL  M     9301 Eureka Lane       NULL
NULL  Terri Lee   Duffy  0      1971-08-01  S     NULL  F     9301 Eureka Lane       NULL
NULL  Roberto     NULL  Tamburello  0     1974-11-12  M     NULL  M     9301 Eureka Lane       NULL
NULL  Rob   NULL  Walters 0     1974-12-23  S     NULL  M     9301 Eureka Lane       NULL
NULL  Ken   J     Sánchez        0     1969-01-29  S     NULL  M     4215 Arleda Lane       NULL
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Real Time Projects Query:

```
select Production.ProductCostHistory.ProductID,Production.ProductCostHistory.ModifiedDate,
Sales.SalesTerritoryHistory.ModifiedDate,Production.ProductDocument.ModifiedDate,
Production.ProductInventory.ProductID,Production.ProductInventory.ModifiedDate,
Production.ProductListPriceHistory.ProductID,Production.ProductListPriceHistory.ModifiedDate,
Sales.SpecialOffer.ModifiedDate from Production.ProductCostHistory as p
inner join Sales.SpecialOffer as s on p.ModifiedDate=s.ModifiedDate


select Production.ProductCostHistory.ProductID,Production.ProductCostHistory.ModifiedDate,
Sales.SalesTerritoryHistory.ModifiedDate,Production.ProductDocument.ModifiedDate
from Production.ProductCostHistory
inner join Sales.SalesTerritoryHistory on Production.ProductCostHistory.ModifiedDate=
Sales.SalesTerritoryHistory.ModifiedDate

--ProductCostHistory.ModifiedDate
=============================================================================================================================
select Production.ProductCostHistory.ProductID,Production.Product.ProductID,
Production.ProductInventory.ProductID,Production.ProductDocument.ProductID,
```

```
Production.ProductListPriceHistory.ProductID,Production.ProductPhoto.ProductID,
from Production.ProductCostHistory
inner join Production.ProductInventory on Production.ProductCostHistory.ProductID=
Production.ProductInventory.ProductID

full outer join Production.ProductCostHistory on Production.ProductCostHistory.ModifiedDate=
Production.ProductListPriceHistory.ModifiedDate
=============================================================================================================================

SELECT A.TABLE_SCHEMA, B.TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.TABLES AS A
INNER JOIN INFORMATION_SCHEMA.COLUMNS AS B
ON A.TABLE_NAME = B.TABLE_NAME AND A.TABLE_SCHEMA = B.TABLE_SCHEMA
WHERE COLUMN_NAME LIKE '%rowguid%' AND TABLE_TYPE = 'BASE TABLE';

select distinct Production.ProductCostHistory.ProductID,Production.ProductCostHistory.ModifiedDate,
Production.Product.ProductNumber,Production.Product.MakeFlag,Production.Product.FinishedGoodsFlag,
Production.Product.Color,Production.Product.SafetyStockLevel,Production.Product.ReorderPoint,
Production.Product.StandardCost,Production.Product.ListPrice,Production.Product.SizeUnitMeasureCode,
Production.Product.Size,Production.Product.WeightUnitMeasureCode,Production.Product.WeightUnitMeasureCode,
Production.Product.Weight,Production.Product.ProductModelID,
Production.ProductInventory.ProductID,Production.ProductInventory.ModifiedDate,Production.Product.Name,
Production.Product.rowguid,Production.WorkOrder.StockedQty
from Production.ProductCostHistory
inner join Production.ProductInventory on Production.ProductCostHistory.ProductID=
Production.ProductInventory.ProductID
inner join Production.Product on Production.ProductCostHistory.ProductID=
Production.Product.ProductID
inner join Production.ProductProductPhoto on Production.ProductCostHistory.ProductID=
Production.ProductProductPhoto.ProductID
inner join Sales.SpecialOfferProduct on Production.ProductProductPhoto.ProductID=
Sales.SpecialOfferProduct.ProductID
inner join Production.TransactionHistory on Sales.SpecialOfferProduct.ProductID=
Production.TransactionHistory.ProductID
inner join Production.WorkOrder on Production.TransactionHistory.ProductID=
Production.WorkOrder.ProductID
inner join Production.ProductListPriceHistory on Production.WorkOrder.ProductID=
Production.ProductCostHistory.ProductID

full outer join Sales.SalesTerritory on Production.Product.rowguid=Sales.SalesTerritory.rowguid
full outer join Sales.SalesTerritoryHistory on Sales.SalesTerritory.rowguid=Sales.SalesTerritoryHistory.rowguid
full outer join Production.ProductCategory on Sales.SalesTerritoryHistory.rowguid=Production.ProductCategory.rowguid
full outer join Purchasing.ShipMethod on Production.ProductCategory.rowguid=Purchasing.ShipMethod.rowguid
full outer join Production.ProductDescription on Purchasing.ShipMethod.rowguid=Production.ProductDescription.rowguid
full outer join Person.Address on Production.ProductDescription.rowguid=Person.Address.rowguid
full outer join Person.AddressType on Person.Address.rowguid=Person.AddressType.rowguid
full outer join Person.StateProvince on Person.AddressType.rowguid=Person.StateProvince.rowguid

full outer join Person.Person on Production.ProductCostHistory.ModifiedDate=Person.Person.ModifiedDate
full outer join Production.ProductSubcategory on Person.Person.ModifiedDate=Production.ProductSubcategory.ModifiedDate
full outer join Production.ProductModel on Person.Person.ModifiedDate=Production.ProductModel.ModifiedDate
full outer join Production.Illustration on Production.ProductModel.ModifiedDate=Production.Illustration.ModifiedDate
full outer join Production.ProductModelProductDescriptionCulture on Production.Illustration.ModifiedDate=
Production.ProductModelProductDescriptionCulture.ModifiedDate
=============================================================================================================================
SELECT DISTINCT
   pch.ProductID,
   pch.ModifiedDate,
   p.ProductNumber,
   p.MakeFlag,
   p.FinishedGoodsFlag,
   p.Color,
   p.SafetyStockLevel,
   p.ReorderPoint,
   p.StandardCost,
   p.ListPrice,
   p.SizeUnitMeasureCode,
   p.Size,
   p.WeightUnitMeasureCode,
   p.Weight,
   p.ProductModelID,
   pi.ModifiedDate,
   p.Name,
   p.rowguid
FROM Production.ProductCostHistory AS pch
full JOIN Production.ProductInventory AS pi ON pch.ProductID = pi.ProductID
full JOIN Production.Product AS p ON pch.ProductID = p.ProductID
full JOIN Production.WorkOrder AS wo ON pch.ProductID = wo.ProductID
```

```
full JOIN Sales.SpecialOfferProduct AS sop ON pch.ProductID = sop.ProductID
full JOIN Production.TransactionHistory AS th ON sop.ProductID = th.ProductID
full JOIN Sales.SalesTerritory AS st ON p.rowguid = st.rowguid
full JOIN Sales.SalesTerritoryHistory AS sth ON st.rowguid = sth.rowguid
full JOIN Production.ProductCategory AS pc ON sth.rowguid = pc.rowguid
full JOIN Purchasing.ShipMethod AS sm ON pc.rowguid = sm.rowguid
full JOIN Production.ProductDescription AS pd ON sm.rowguid = pd.rowguid
full JOIN Person.Address AS a ON pd.rowguid = a.rowguid
full JOIN Person.AddressType AS at ON a.rowguid = at.rowguid
full JOIN Person.StateProvince AS sp ON at.rowguid = sp.rowguid
full JOIN Person.Person AS per ON pch.ModifiedDate = per.ModifiedDate
full JOIN Production.ProductSubcategory AS psc ON per.ModifiedDate = psc.ModifiedDate
full JOIN Production.ProductModel AS pm ON per.ModifiedDate = pm.ModifiedDate
full JOIN Production.Illustration AS i ON pm.ModifiedDate = i.ModifiedDate
full JOIN Production.ProductModelProductDescriptionCulture AS pmc ON i.ModifiedDate = pmc.ModifiedDate;

================================================================================================================================
```