**IMPORTING SQL**

===========================================================

from pyspark import SQLContext

sqlContext = SQLContext(sc)

departmentsJson = sqlContext.jsonFile("/user/cloudera/pyspark/departments.json")

departmentsJson.registerTempTable("departmentsTable")

departmentsData = sqlContext.sql("select \* from departmentsTable")

for rec in departmentsData.collect():

print(rec)

===========================================================

**SQL FROM PYSPARK**

----------------------------------------------------------------------------------------------------

from pyspark.sql import HiveContext

sqlContext = HiveContext(sc)

depts = sqlContext.sql("select \* from departments")

for rec in depts.collect():

print(rec)

sqlContext.sql("create table departmentsSpark as select \* from departments")

depts = sqlContext.sql("select \* from departmentsSpark")

for rec in depts.collect():

print(rec)

**# Using Hive**

===========================================================

from pyspark.sql import HiveContext

sqlContext = HiveContext(sc)

sqlContext.sql("set spark.sql.shuffle.partitions=10");

joinAggData = sqlContext.sql("select o.order\_date, round(sum(oi.order\_item\_subtotal), 2), \

count(distinct o.order\_id) from orders o join order\_items oi \

on o.order\_id = oi.order\_item\_order\_id \

group by o.order\_date order by o.order\_date")

for data in joinAggData.collect():

print(data)

===========================================================

**# Using Pyspark native sql**

----------------------------------------------------------------------------------------------------

from pyspark.sql import SQLContext, Row

sqlContext = SQLContext(sc)

sqlContext.sql("set spark.sql.shuffle.partitions=10");

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

ordersMap = ordersRDD.map(lambda o: o.split(","))

orders = ordersMap.map(lambda o: Row(order\_id=int(o[0]), order\_date=o[1], \

order\_customer\_id=int(o[2]), order\_status=o[3]))

ordersSchema = sqlContext.inferSchema(orders)

ordersSchema.registerTempTable("orders")

orderItemsRDD = sc.textFile("/user/cloudera/sqoop\_import/order\_items")

orderItemsMap = orderItemsRDD.map(lambda oi: oi.split(","))

orderItems = orderItemsMap.map(lambda oi: Row(order\_item\_id=int(oi[0]), order\_item\_order\_id=int(oi[1]), \

order\_item\_product\_id=int(oi[2]), order\_item\_quantity=int(oi[3]), order\_item\_subtotal=float(oi[4]), \

order\_item\_product\_price=float(oi[5])))

orderItemsSchema = sqlContext.inferSchema(orderItems)

orderItemsSchema.registerTempTable("order\_items")

joinAggData = sqlContext.sql("select o.order\_date, sum(oi.order\_item\_subtotal), \

count(distinct o.order\_id) from orders o join order\_items oi \

on o.order\_id = oi.order\_item\_order\_id \

group by o.order\_date order by o.order\_date")

for data in joinAggData.collect():

print(data)

===========================================================

**SORTING IN SQL**

----------------------------------------------------------------------------------------------------

**#Global sorting and ranking**

select \* from products order by product\_price desc;

select \* from products order by product\_price desc limit 10;

**#By key sorting**

#Using order by is not efficient, it serializes

select \* from products order by product\_category\_id, product\_price desc;

**#Using distribute by sort by (to distribute sorting and scale it up)**

select \* from products distribute by product\_category\_id sort by product\_price desc;

**#By key ranking (in Hive we can use windowing/analytic functions)**

select \* from (select p.\*,

dense\_rank() over (partition by product\_category\_id order by product\_price desc) dr

from products p

distribute by product\_category\_id) q

===========================================================

**GET NUMBER OF CANCLED ORDERS IN MOTH OF JANUVARY IN 2014**

select order\_status, count(1) from orders

where from\_unixtime(cast(substr(order\_date, 1, 10) as int)) like '2014-01%' group by order\_status;

where dr <= 2 order by product\_category\_id, dr;