ISIT312 Big Data Management
SIM S2 2020
Assignment 3

All files left on Moodle in a state "Draft(not submitted)" will not be evaluated. Please refer to the submission dropbox on Moodle for the submission due date and time.

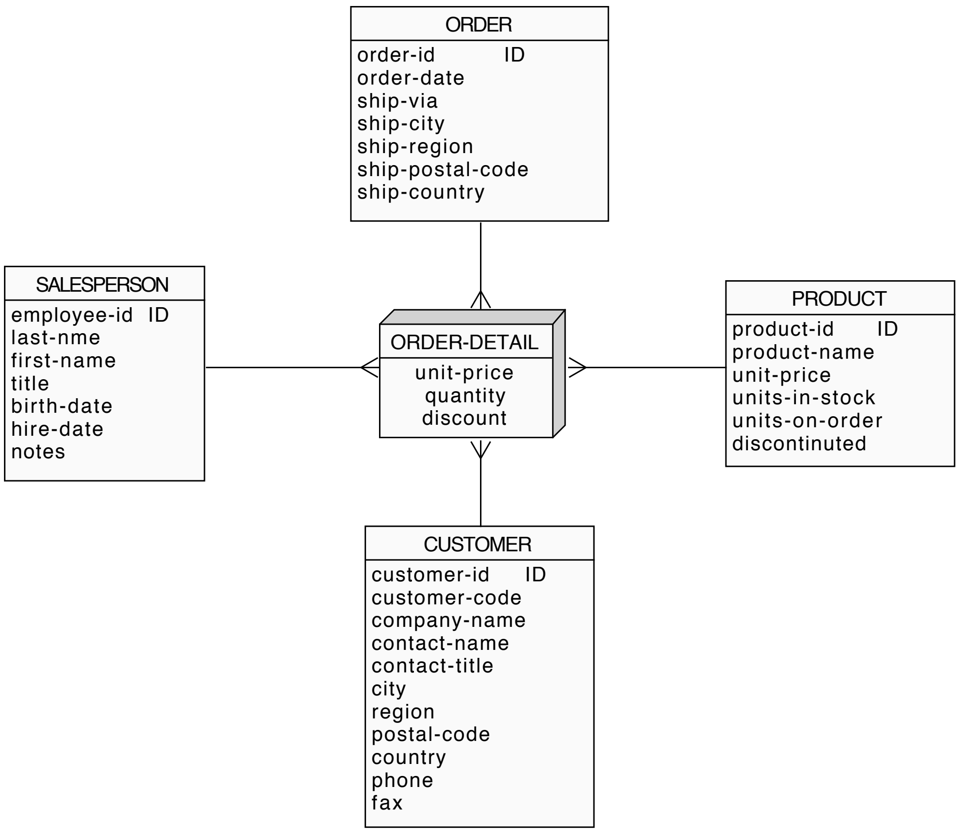
This assignment contributes to **20%** of the total evaluation in the subject. This assignment consists of 4 tasks. Specification of each task starts from a new page.

Task 1. Apache Pig (8 marks)
Task 2. Spark and HBase (4 marks)
Task 3. Spark and Hive (5 marks)
Task 4. Spark Streaming (3 marks)

All the tasks must be completed in the "BigDataVM" virtual machine used in ISIT312.

It is a requirement that all Laboratory and Assignment tasks in this subject must be solved individually without any cooperation with the other students. If you have any doubts, questions, etc. please consult your lecturer or tutor during lab classes or office hours. Plagiarism will result in a FAIL grade being recorded for that assessment task.

Task 1. Data Queries in Pig Latin (8 marks)



Consider the above conceptual schema of a data warehouse. The data of the schema is stored in the files `customer.tbl`, `order_details.tbl`, `order.tbl`, `product.tbl` and `salesperson.tbl`, all of which are available in a "Resources" folder of Assignment 3 on Moodle.

Note that each file has a header with information about the meanings of data in each column. A header is not a data component of each file. Remove the headers before transferring the files to HDFS

For each of the following queries on the data warehouse, implement and execute a script in Pig Latin to return the correct result:

(1) Find the *number of orders* whose ship-city is London.
(2) Find the *number of products* that were *not* ordered in 1996.
(3) Find the *order value (i.e., unit price multiplied by quantity of products per order)* for order IDs 10270 to 10279.
(4) Sort the salespersons by *the total order value of orders they handled* in a descending order, and find the employee ID, first name and last name of the *top three* salespersons.
(5) Find the *number of orders* for products (i.e., product names) "Ikura" or "Tofu".

Deliverables
Produce a report `solution1.pdf` which clearly documents the Pig scripts and output. Submit the report.

Task 2. Spark and HBase (4 marks)

The file **`1902`** is a weather record dataset collected from one station in U.S. in 1902. Each record is a line in the ASCII format. The following shows one sample line with some of the salient fields annotated.

```
0057
332130      # USAF weather station identifier
99999       # WBAN weather station identifier
19500101    # observation year, month and date
0300        # observation time
4
+51317      # latitude (degrees x 1000)
+028783     # longitude (degrees x 1000)
FM-12
+0171       # elevation (meters)
99999
V020
320         # wind direction (degrees)
1   # quality code
N
0072
1
00450       # sky ceiling height (meters)
1   # quality code
C N
010000      # visibility distance (meters)
1   # quality code
N 9
-0128       # air temperature (degrees Celsius x 10)
1   # quality code
-0139       # dew point temperature (degrees Celsius x 10)
1   # quality code
10268       # atmospheric pressure (hectopascals x 10)
1   # quality code
```

This file is available in a "Resources" folder of Assignment 3 on Moodle. The *objective* of this task is to extract useful information from the file in Spark-shell, perform basic aggregations and save the data into HBase.

Create a script of Scala sourcecode in a text editor (such as gedit) which implements the following Spark-shell functions:

(1) Create a DataFrame named `weatherDF` based on **`1092`** with the following fields:

<recordID: String> *# the first 25 characters as a record identifier*
<station: String> *# USAF weather station identifier*
<month: String>
<date: String>
<hour: String>
<temperature: Double> # air temperature

(2) Compute (and return) the <u>maximum, minimum and average temperatures</u> for each month in `weatherDF`. (You can use either DataFrame operations or SQL statements.)

(3) Convert the first 10 rows in `weatherDF` into a HBase table named `weatherHB` by using the Hortonworks Spark-HBase-Connector (shc), namely using package numbers from `org.apache.spark.sql.execution.datasources.hbase._`
The table `weatherHB` uses recordID as rowkeys and has 3 column families. The first column family includes the station column, the second includes the month, date and hour columns, and the third includes the temperature column.
(Hint: for convenience you can cast the temperature column to `StringType` before saving the data to HBase.)

After creating the Scala script, start Spark-shell with the Hortonworks Spark-HBase-Connector (shc), namely,

```
$SPARK_HOME/bin/spark-shell --master local --packages
com.hortonworks:shc-core:1.1.0-2.1-s_2.11 --repositories
http://repo.hortonworks.com/content/groups/public/
```

Start HBase master. Run the Scala script in Spark-shell using `:paste` command.

Lastly, show `weatherHB` has been created by using `scan` command in HBase shell. *(Note. you should not use Zeppelin in this task.)*

Deliverables
Name the Scala script as `solution2-script.txt`. Summarise all the Bash and HBase input and output in a PDF report named `solution2-report.pdf`. The Scala script must be executable in Spark-shell. The PDF report must demonstrate your operation and result of this task. Submit both files.

Task 3. Spark and Hive (5 marks)

DATASETS: **`apat63_99.txt`** and **`cite75_99.txt`** available in the *datasets* folder of Desktop of the BigDataVM virtual machine for ISIT312/ISIT912.
The source of datasets is http://www.nber.org/patents/.
The dataset **`apat63_99.txt`**, which was used in Task 3 of Assignment 1, contains about 3 million records for the U.S. patents. A description of this dataset was provided in the specification of Assignment 1. The accompanied dataset **`cite75_99.txt`** contains more than 16 million lines of citation records. The following are the first few lines:

```
"CITING","CITED"
3858241,956203
3858241,1324234
3858241,3398406
3858241,3557384
3858241,3634889
3858242,1515701
3858242,3319261
3858242,3668705
3858242,3707004
...
```

For example, the second line shows that patent 3858241 cites patent 956203. The file is sorted by the citing patent. A *citation count* of a patient refers to the number of other patents that cite this patent. For example, if there are 100 patients citing patient X, then the citation count of patent X is 100.

Implement a *self-contained* application in Scala that returns *the top 20 most cited patents whose first inventors are from Australia*. This Scale code should include the following steps: (i) Filter the patient records whose main investors are Australian; (ii) aggregate the citation records to produce the citation counts for patients; (iii) perform an inner join between two set of records and sort the records according to the citation counts.

The final output is saved to Hive as a table named `HiTAB` that contains two columns, one for the patent IDs and the other for the citation counts. The connection of Spark to Hive must use the metastore service of Hive.

Use the following sample code for your application:

```
import org.apache.spark.sql.{SaveMode, SparkSession}
import org.apache.spark.sql.functions.{col, count, desc}

object solution3 {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder.appName("task4")
      .config("spark.master", "local")
      .config("hive.metastore.uris", "thrift://localhost:9083")
      .config("hive.metastore.schema.verification", "true")
      .enableHiveSupport()
      .getOrCreate()


     spark.sparkContext.setLogLevel("ERROR")
```

```
        /*
        * your code …
        */

        spark.stop()
      }
    }
```

Compile your completed code with **`scalac`** and submit the application with `$SPARK_HOME/bin/spark-submit --master local[*].`

Show `HiTAB` in the Hive shell or Zeppelin using the `show table HiTAB` command.

Deliverables

Implement a self-contained Scala sourcecode `solution3.scala`. This sourcecode must be compilable. Produce a PDF report `solution3-report.pdf` that summarises all Bash and Hive input and output, which demonstrate your operations and result for this task. Submit the Scala sourcecode and the PDF report.

Task 4. Spark Streaming (4 marks)

DATASET: A file named **`by-day`** in the "Resource" folder on the Moodle site. The file includes some daily retail records of a retailer in one year.

Based on the following sample Scala code, implement a *streaming query* on the dataset. The input source of this query is HDFS. Download (and unzip) the files, and transfer the files to HDFS.

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.streaming.{Trigger,ProcessingTime}

val retail_data = ... <specify your path to data source>
val staticDataFrame = spark.read.format("csv")
      .option("header", "true")
      .option("inferSchema", "true")
      .load(retail_data)

val staticSchema = staticDataFrame.schema
staticDataFrame.printSchema()
// root
// |-- InvoiceNo: string (nullable = true)
// |-- StockCode: string (nullable = true)
// |-- Description: string (nullable = true)
// |-- Quantity: integer (nullable = true)
// |-- InvoiceDate: timestamp (nullable = true)
// |-- UnitPrice: double (nullable = true)
// |-- CustomerID: double (nullable = true)
// |-- Country: string (nullable = true)

spark.conf.set("spark.sql.shuffle.partitions", 2)

val streamingDataFrame = spark.readStream
      .schema(staticSchema)
      .option("maxFilesPerTrigger", 10)
      .format("csv")
      .option("header", "true")
      .load(retail_data)

streamingDataFrame.isStreaming //true if streaming

val purchaseQuery = streamingDataFrame
      /* <to be completed>
      */

val query = purchaseQuery
      .writeStream
      .format("console")
      .queryName("customer_purchases")
      .outputMode("complete")
      .trigger(ProcessingTime("5 seconds"))
      .start()
```

The streaming query performs the following functions:

- It filters out records with missing StockCode values (i.e., with a null value in the StockCode column), and
- It returns the average Quantity per (non-null) StockCode in a descending order. (Note: This average quantity is computed up to the current moment in the streaming query, and thus it is updated each time a query result is returned.)

Implement the Scala sourcecode of this streaming query in a script named `solution4-scala.txt` with a text editor (e.g., gedit).

Run to script in Spark-shell using `:paste` command. Report the first 20 rows of the initial five return batches. *(Note. you should not use Zeppelin in this task.)*

Deliverables
A Scala script `solution4-scala.txt` and a PDF report `solution4-report.pdf`. The script must be executable in Spark-shell. The PDF report must include your operation commands and the return batches. Submit the Scala script and the report.

End of specification