Big Data Systems

Assignment, 2020

Hadoop, HDFS and Hive

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Table of Contents

Assignment Overview 2

General Instructions 2

Useful Software 2

Additional Material 2

Purpose of the assignment 2

Part 1: Theoretical questions 3

Part 2: Practical questions 5

Introduction 5

Questions/Tasks 6

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Assignment Overview

**Areas covered:**

* Hadoop Distributed File System (HDFS).
* Apache Hive.

**Submission Deadline: 20/04/2020 23:59**

## General Instructions

The current assignment includes two parts of mandatory questions and tasks of 118 points, in total. The highest score that can be achieved in this assignment is 100 points (i.e., 18 points are considered as bonus points).

Note that although a certain dataset is provided and can be used to answer/complete the question/tasks of this assignment, it is not required to successfully complete the assignment. The questions could be answered even without using the dataset. However, it is recommended to use it (i.e., try to build the code requested over the given dataset, where it is feasible) in order to understand better what is asked, as well as practice over the areas considered in this assignment. The dataset is only used for practicing in Part 2.

***Deliverables***

*Please provide your answers into a single TXT file, with the corresponding question-numbers before each answer (in order to be clear which is the question of each answer).*

## Useful Software

Use the Cloudera Quickstart (CDH, for short) to practice on the tasks of this assignment (for installation details see slides in SWInstallation.pdf uploaded in the space of the course). Note that the services (from CDH) that could be used for completing this assignment are limited to the following:

* Apache Hadoop (including MapReduce and HDFS),
* Apache Hive, and
* Hue

## Additional Material

* The given dataset (orders\_dataset.zip) includes the following files:
  + order\_details.csv: Details about orders.
  + orders.csv: Information about orders.
  + rates.zip: historical conversion rates to EUR.

## Purpose of the assignment

The purpose of this assignment is to:

1. Understand the main concepts and approaches discussed during the corresponding lectures (HDFS, storage formats and Hive), and
2. Practice over Apache Hive and HDFS.

# Part 1: Theoretical questions

<40 points>

Q1.1 <4pt.>: Which of the following issues may be caused by lot of small files in HDFS:

1. NameNode memory usage increases significantly.
2. Overall network load increases.
3. It is required more cluster nodes to process the data.
4. I/O rate will be faster.

Select the correct options (one or more) ***and explain*** your answer, **shortly**.

Q1.2 <4pt.>: How is uniform data distribution across the servers achieved in HDFS?

1. By splitting files into blocks
2. By replication

Select one of the aforementioned options ***and explain*** your answer, **shortly**.

Q1.3 <4pt.>: If you have a very important file and you aim to store it into HDFS, what is the best way to minimize the risk from losing it?

Q1.4 <4pt.>: You were told that two servers in HDFS were down: Datanode and Namenode. Which would be your reaction:

* 1. It’s OK, replication factor is 3.
  2. Restore Datanode first.
  3. Restore Namenode first.

Select one of the aforementioned options ***and explain*** your answer, **shortly**.

Q1.5 <4pt.>: You are writing a 10GB file into HDFS with a replication of 2 and block size of 64MB. How much **total** disk space will this file use? **E*xplain*** your answer, **shortly**.

Q1.6 <4pt.>: Your colleague mistakenly dropped the table 'access\_log' created with the following statement, in Hive:

CREATE EXTERNAL TABLE access\_log (

id STRING,

user\_id STRING,

request STRING,

response STRING,

status\_code INT

)

LOCATION '/data/access\_logs'

STORED AS TEXTFILE;

Which would be your reaction? **Explain** your answer**, shortly**.

Q1.7 <4pt.>: You have a bunch of data in your local filesystem (which is **not** part of the cluster) and you need to load it in Hive. Describe the steps that you are going to follow in order the data to be accessible through Hive queries. *Code is not required, just a short description of the steps.*

Q1.8 <4pt.>: You have an external table in Hive and want to store this data in more compact and efficient format. Your decision is:

1. Create a table in a new format with the CREATE TABLE statement and fill it from the external table with the INSERT INTO TABLE statement.
2. CREATE VIEW over an external table.
3. Create and fill a new table with the CREATE TABLE ... AS SELECT statement.

Select one of the options ***and explain*** your answer, **shortly**.

**In addition**, provide at least one type of format that could improve the performance on query aggregations. **E*xplain*** your answer, **shortly**.

Q1.9 <4pt.>: Why does partitioning optimize Hive queries? Please provide a **short** answer.

Q1.10 <4pt.>: You join two Hive tables: A(key INT, value STRING) and B(key INT, value STRING). Тable B is small enough to be stored in RAM of a single compute node in the cluster and A is much bigger than B (A exceeds the average RAM of a cluster node). Provide an efficient query optimizing the join (INNER JOIN over A.key=B.key) between A and B, in Hive. Assume that the query returns A.value and B.value.

# Part 2: Practical questions

<60 points>

## Introduction

The purpose of this part is to apply transformations and analysis tasks over the data received through files. We assume that you receive the data related to orders of an international company with multiple offices. Based on this data, you initially aim to re-structure the data into a more efficient structure, and then apply further analysis. Please find below details about the files format and data considered as input.

**Main Orders File (file name: orders.csv):**

|  |  |
| --- | --- |
| **Field** | **Description** |
| ORDER\_NUMBER | Order unique identifier |
| ORDER\_DATE | Date of Order |
| SHIPPED\_DATE | Date the order shipped |
| STATUS | Order status |
| COMMENTS | Comments over the order |
| CUSTOMER\_NUMBER | Customer unique identifier |
| CUSTOMER\_NAME | Customer name |
| CUST\_CITY | City of customer |
| CUST\_STATE | State of customer |
| CUST\_COUNTRY | Country of customer |
| CUST\_COUNTRY\_ISO | ISO code of customer country |
| SALES\_CURRENCY | Currency of sales price |
| SALES\_REP\_ID | Sales representative id |
| SALES\_REP\_FIRSTNAME | First name of Sales representative |
| SALES\_REP\_LASTNAME | Surname of Sales representative |
| OFFICE\_CODE | Code of office the sales representative belongs to |
| REPORTING\_PATH | Sales representative reporting path |
| OFFICE\_CITY | City of office |
| OFFICE\_STATE | State of office |
| OFFICE\_TERRITORY | Territory of office |
| OFFICE\_COUNTRY | Office country |

**Order Details (file name: order\_details.csv):**

|  |  |
| --- | --- |
| **Field** | **Description** |
| ORDER\_NUMBER | Order unique identifier |
| PRODUCT\_CODE | Unique identifier of the product |
| PRODUCT\_NAME | Name of the product |
| PRODUCT\_CATEGORY | Category of the product |
| PRODUCT\_VENDOR | Vendor of the product |
| QUANTITY\_IN\_STOCK | Quantity in stock |
| BUY\_PRICE | Purchase price, per item |
| QUANTITY\_ORDERED | Number of items ordered, for the specific product |
| UNIT\_PRICE | Price of the product (per item) |

Note the following:

* The Sales currency included in the orders file applies to all the corresponding sales prices (UNIT\_PRICE) in the order\_details file. We consider that the currency of BUY\_PRICE is always EUR.
* REPORTING\_PATH refers to organizational structure of the sales department. i.e., the SALES\_REP\_ID of the line manager is the first element of the list, his/her line manager is included as second element, etc. The last element of the list is always -1, indicating that there is no line manager of the head of department.

**Currency conversion rates (from any currency to EUR):**

* Files included in zip file rates.zip.
* Each file includes the conversion rates for each date, in JSON format.
* The date included in both the file name and an element with key “date”.
* All the rates included in sub-element named “rates”.

## Questions/Tasks

Q2.1 <7pt.>: Provide the statements in order to

a) Create the following folder structure within the HDFS folder “*cloudera*” located in the path */user/cloudera.*

* The folder RawData is included in the cloudera folder.
* The folder OrdersData is included in the RawData folder.
* The folder OrderDetailsData is included in the RawData folder.

b) Load the following data files into the corresponding HDFS folder*.*

|  |  |
| --- | --- |
| **Data type** | **Folder** |
| orders.csv | Load the data into the folder OrdersData |
| order\_details.csv | Load the data into the folder OrderDetailsData |
| Exchange rates files included in the rates folder in rates.zip file. | Load the data into the folder Rates which should be included in the folder RawData |

*Hints/Clarifications:*

* *Consider that the aforementioned files have already been uploaded into a node having HDFS client set-up properly.*
* *To load into hdfs a set of files, you could either load the folder and then rename it, or load it and rename in a single command.*

Q2.2 <8pt.>: Provide the statements in order to

- Create a new database with name “orders\_data”

- Make the following data accessible through Hive (i.e., the data could be queried through Hive).

* orders.csv
* order\_details.csv
* Exchange rates

The tables that are going to be created should belong to the database “orders\_data”. Tables-Create statements, as well as the statements used to load the data into the tables (either HDFS statements or Hive statements) should be included in the answer of this question.

*Hints:*

* *There are multiple approaches/statements to load the data; each of them is accepted as answer. Suggested: use of external tables.*
* *Use a complex data type for defining the column REPORTING\_PATH.*
* *Identify and define correctly the field and collection delimiters.*
* *In external tables, use STRING data types (instead of VARCHAR or CHAR) as a data type to define textual fields.*
* *Do not forget to ignore the headers.*
  + *Use the following property to do so: tblproperties ("skip.header.line.count"="1")*
  + *See also examples provided for additional details.*
* *In order to make accessible the exchange rates through Hive, create a simple table that has a single string column. A similar approach has been given in the DW example.*

Q2.3 <8pt.>: Provide the statements in order to create the following partitioned table in the database “orders\_data” (created in Q2.2):

* Name of the partitioned table: stg\_orders
* Storage format: text file
* Fields delimiter: symbol “|”
* Delimiter of collection items: “,”
* Partitioned by the year of the ORDER\_DATE

The table stg\_orders aims to store the historical orders data in its raw format (similar to the one used to load the data in the previous question). Then, create three partitions one for each of the following years: 2003, 2004, 2005. Finally, load the orders data into the corresponding partitions such that:

* Assuming that the stg\_orders has already data for year 2003, the data for order date in the year 2003 should be updated with the new data
* The data for order date in the years 2004 and 2005 should be appended.

Provide all the statements used to apply the operations described above.

*Hints:*

* *Load the data into the table stg\_orders using the table storing the data in orders.csv (select from this table the corresponding data), created in the question Q2.2.*
* *To extract the year from either a TIMESTAMP or DATE column [COLUMN] you use the function: year([COLUMN]).*
* *To convert either a TIMESTAMP or a STRING column [COLUMN] into a DATE column you use the function cast([COLUMN] as date)*
* *There are two approaches for inserting the data into the corresponding partitions: static and dynamic. The static is given by explicitly specifying the partition the data is stored (see the basic examples provided), while in the dynamic (see an example in DW example), Hive is responsible to insert the data into the corresponding partitions. Both approaches are acceptable. To apply the dynamic partitioning, use the following settings before you run the insert statement. In addition, you need to add the partition field as the last field in the selection clause.*
  + *set hive.exec.dynamic.partition=true;*
  + *set hive.exec.dynamic.partition.mode=nonstrict;*

Q2.4 <8pt.>: Consider the following staging tables storing the historical, raw records for order\_details and rates, respectively:

CREATE TABLE IF NOT EXISTS orders\_data.stg\_order\_details (

ORDER\_NUMBER BIGINT,

PRODUCT\_CODE STRING,

PRODUCT\_NAME STRING,

PRODUCT\_CATEGORY STRING,

PRODUCT\_VENDOR STRING,

QUANTITY\_IN\_STOCK INT,

BUY\_PRICE DECIMAL,

QUANTITY\_ORDERED INT,

UNIT\_PRICE DECIMAL

)

STORED AS orc;

CREATE TABLE IF NOT EXISTS orders\_data.stg\_conv\_rates (

INDATE DATE,

RATES STRING

)

STORED AS orc;

Consider the tables delta\_order\_details and delta\_rates created in Q2.2 and storing the raw order\_details and rates data, respectively. If we consider all the columns in the tables delta\_order\_details and delta\_rates are given in STRING data type, provide the statements in order to insert (not overwrite) the data from the tables delta\_order\_details and delta\_rates into the stg\_order\_details and stg\_conv\_rates, respectively, by converting the columns into the proper data type.

*Hints:*

* *Consider that the external table delta\_rates has a single STRING column.*
* *Consider that each record of the external table delta\_rates has all the rates of a certain date.*
* *To convert a STRING column [COLUMN] into a column of data type [DATA\_TYPE], use the function cast([COLUMN] as [DATA\_TYPE]). Note that the valid conversions are described in Slide 25.*
* *In order to parse a json string field, use the function get\_json\_object (see examples in Slide 59). The format of the function is given as follows:*
  + *get\_json\_object([JSON\_STRING], [JSON\_PATH]), where JSON\_PATH the path of the field to extract (e.g., $.date for extracting the date and $.rates for extracting the rates). See also an example in DW Example.*
  + *Once you extract the date field from json apply a casting operator to convert it into date.*

Q2.5 <8pt.>: Provide the statements in order to build the following dimensions (dim\_status and dim\_product) from the orders and order\_details, respectively.

CREATE TABLE orders\_data.dim\_status (

STATUS\_KEY INT,

STATUS STRING

)

STORED AS orc;

CREATE TABLE orders\_data.dim\_product (

PRODUCT\_KEY INT,

PRODUCT\_CODE STRING,

PRODUCT\_NAME STRING,

PRODUCT\_CATEGORY STRING,

PRODUCT\_VENDOR STRING

)

STORED AS orc;

Note that the STATUS\_KEY and PRODUCT\_KEY are the surrogate keys that should be generated, and uniquely identify the unique values of the corresponding records (e.g., each unique value of STATUS should be assigned to a unique STATUS\_KEY).

*Hint:*

* *Provide the queries over either the tables* stg\_order\_details (Q2.4), stg\_orders (Q2.3), or the tables defined in Q2.2.
* *You could work as follows: (1) create a temporary table storing the unique values, and (2) query the temporary table (using the windows function row\_number() OVER () in order to assign a unique id/surrogate key) and store the result to the corresponding dimension table. To create the temporary table use CTAS, like:*
  + *CREATE TEMPORARY TABLE [TABLE\_NAME] as [QUERY\_EXPRESSION]*

Q2.6 <9pt.>: Provide the statements in order to build the following fact table over the data stored in thetables orders\_data.stg\_order\_details and orders\_data.stg\_orders. Use also the following dimensions defined in the previous task (Q2.5):

* orders\_data.dim\_status
* orders\_data.dim\_product

Definition of fact table:

CREATE TABLE orders\_data.fact\_orders (

ORDER\_DATE\_KEY INT, -- Surrogate key of time dimension for ORDER\_DATE

SHIPPED\_DATE\_KEY INT, -- Surrogate key of time dimension for SHIPPED\_DATE

STATUS\_KEY INT, -- Surrogate key of status dimension for STATUS

PRODUCT\_KEY INT, -- Surrogate key of product dimension for PRODUCT\_CODE

NUMBER\_OF\_ORDERS INT, -- Number of distinct orders

TOTAL\_QUANTITY\_ORDERED INT, -- Total number of QUANTITY\_ORDERED

SALES\_AMOUNT DECIMAL(15,2), -- Total amount paid by the customer

MAX\_UNIT\_PRICE DECIMAL(15,2), -- Maximum unit price

AVG\_UNIT\_PRICE DECIMAL(15,2), -- Average unit price

MIN\_UNIT\_PRICE DECIMAL(15,2), -- Minimum unit price

TOTAL\_PURCHASE\_AMOUNT DECIMAL(15,2) -- Total amount paid by the company

)

STORED AS orc;

To build the dimension date (orders\_data.dim\_date) that is required in order to find the ORDER\_DATE\_KEY and SHIPPED\_DATE\_KEY, use the following scripts.

-- Create the date dimension table

DROP TABLE IF EXISTS orders\_data.dim\_date;

CREATE TABLE IF NOT EXISTS orders\_data.dim\_date (

date\_id int,

date string,

year int,

month int

)

STORED AS orc;

-- Create the records for each date included in order\_date and shipped\_date

with min\_max\_dates as (

select cast('2003-01-06' as DATE) as min\_date,

cast('2006-08-23' as DATE) as max\_date

)

INSERT INTO orders\_data.dim\_date

(date\_id, `date`, `year`, `month`)

select

row\_number() over () as date\_id,

date,

year(date) as year,

month(date) as month

from (select

date\_add(t.min\_date, a.pos) as date

from (

select posexplode(split(repeat("o", datediff(max\_date, t.min\_date)), "o"))

from min\_max\_dates t

) a, min\_max\_dates t) d

order by date;

*Hints:*

* *The measures are given for each unique combination of keys.*
* *The calculation of measures SALES\_AMOUNT, MAX\_UNIT\_PRICE, AVG\_UNIT\_PRICE, MIN\_UNIT\_PRICE is based on QUANTITY\_ORDERED, UNIT\_PRICE (i.e., SALES\_AMOUNT = QUANTITY\_ORDERED\*UNIT\_PRICE).*
* *The calculation of the measure TOTAL\_PURCHASE\_AMOUNT is based on the QUANTITY\_ORDERED and BUY\_PRICE (i.e., TOTAL\_PURCHASE\_AMOUNT = QUANTITY\_ORDERED\*BUY\_PRICE).*
* *Both ORDER\_DATE\_KEY INT, and SHIPPED\_DATE\_KEY get values from the dimension table orders\_data.dim\_date (but they might be different dates with different date\_ids).*
* *Suggested approach: Define a temporary table by joining stg\_orders and stg\_order\_details and selecting the following fields: order\_number, order\_date, shipped\_date, status, product\_code, quantity\_ordered, unit\_price, buy\_price. Note that the date fields could be converted to DATE. Then use this temporary table to join with dim tables in order to create the fact records and insert them into the fact table.*

Q2.7 <10pt.>: Consider the table product\_sales created as follows:

-- Create products\_sales table

CREATE TABLE IF NOT EXISTS orders\_data.products\_sales (

ORDER\_DATE DATE ,

PRODUCT\_CODE STRING,

SALES\_AMOUNT DECIMAL(15,2)

)

STORED AS orc;

-- Insert sales amount per order\_date and product\_code into the products\_sales

insert into orders\_data.products\_sales

select cast(o.order\_date as DATE) as ORDER\_DATE,

d.PRODUCT\_CODE,

sum(d.unit\_price\*d.quantity\_ordered) as SALES\_AMOUNT

from orders\_data.stg\_order\_details d, orders\_data.stg\_orders o

where d.order\_number=o.order\_number

group by d.product\_code, cast(o.order\_date as DATE);

Provide the statements answering the following questions:

1. For each product\_code and order\_date, find the SALES\_AMOUNT of the current, the previous and the next day.
2. For each product\_code and order\_date, find the following:
   * the SALES\_AMOUNT, and
   * the difference between the SALES\_AMOUNT of the product and the SALES\_AMOUNT of the product with **maximum** SALES\_AMOUNT in the given order\_date.
3. For each product\_code, find the cumulative sum of SALES\_AMOUNT over the order\_date.

*Hints:*

* *Window functions in Hive:* [*link*](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+WindowingAndAnalytics)*.*
* *In (b), use windows function partitioning by order\_date and ordering properly.*
* *In (c), use windows function with corresponding ordering.*

Q2.8 <20pt.>: In the following, it is requested to find the [association rules](https://en.wikipedia.org/wiki/Association_rule_learning) ([link1](http://infolab.stanford.edu/~ullman/mining/pdf/assoc-rules1.pdf), [link2](http://www.mmds.org/mmds/v2.1/ch06-assocrules.pdf)) of the following form:

* X => Y; i.e., if X is included in an order, the probability of having Y ordered, as well, is C%.

where X,Y are PRODUCT\_CODEs, and C is a given threshold.

Consider also the following:

Support: Supp(**X**)=Number of orders containing all the PRODUCT\_CODEs contained in the set **X.**

Confidence: conf(X=>Y)=supp({X} U {Y})/supp({X}).

In particular, provide the statements in order to find the pairs of products (X,Y), along with their confidence value, for conf(X=>Y) ≤ 80% and supp({X} U {Y}) > 1.

*Hints:*

* *Suggested approach: compute the supp({X}) for all product\_codes over the stg\_order\_details (you could store the results into a temporary table). Compute the supp({X} U {Y}) by joining stg\_order\_details with itself (attension: do not count twice the product-product pairs – to avoid this you could assign a row\_number into the records). Then, the final result is given by joining the previous tables and applying the confidence threshold.*