Data Warehousing 2016-17
Project Assignment

v0

December 15, 2016

WARNING: THIS MIGHT NOT BE THE CURRENT VERSION OF THE ASSIGNMENT, CHECK FOR THE LAST VERSION ON THE AULAWEB MODULE OF THE CLASS

Instructions for installing docker and running the container are included as an appendix to this document. They are the same provided for Hive and Spark SQL proposed activities (i.e., exercise 0).

General rules for project development and subsmission:

- The project must be developed by all the students taking the Data Warehouse course for 9 or 12 ECTS, or integrating previous DW editions passed in previous academic years with additional 6 ECTS.
- The project must be developed by single students or by teams of two students (pairs). The intention to develop the project as a team must be communicated via email to the instructor when starting working on the project.
- The project must be completed and uploaded on Aulaweb at least 48 hours before the oral exam. Students developing the project in teams can sustain the oral exam on different dates.

Projects on different data sources may be possible. In case you have a proposal, directly contact the instructors to check its suitability.

What should a submission include? Each student/team must upload on Aulaweb a zip file containing all the developed code and a pdf file with the required documentation for each of the requests below.
The code must be adequately structured in directories and instructions on how to use the code included in the zip file must also be included.
The documentation must include all the assumptions, conceptual and logical schema adequately commented and motivated, description and motivations of the approaches followed in the various steps, a critical analysis (and comparison when appropriate) on the use of tools/frameworks and relevant outcomes.
Please also include in the documentation a rough estimate of the effort (in terms, e.g., of number of hours) devoted to the project overall, and on the various parts below.

1 Operational data sources inspection and profiling

The operational data source consists in a variant of the Adventureworks OLTP database. It represents a fictitious bicycle parts wholesaler with a hierarchy of nearly 300 employees, 500 products, 20000 customers, and 30000 sales each having an average of 4 line items. In the zip file you find:

- a diagram of the operational database schema (`AdventureWorksDBDiagram.pdf`);
- the content of the database exported in csv, a file per table (csv files included in the directory `AdventureWorksDB`);
- the script `AdventureworksDB.sql`, included in the same directory of the csv files, and that, executed with

  ```
  psql -c "CREATE DATABASE \"Adventureworks\";"
  psql -d Adventureworks < AdventureworksDB.sql
  ```

  (with the csv files in the same directory of the script) creates the schemas and populates the tables.

For data inspection and profiling, besides PostgreSQL queries, you may rely on Trifacta `www.trifacta.com`, just uploading the csv files. Include in the documentation any insight you get from source inspection and profiling.

2 Data warehouse conceptual design

Produce a data warehouse conceptual design according to the DFM notation. Identify the fact(s) of greatest interest for analysis, starting from the analysis of operational sources and making some assumptions (that must be made explicit) about the workload. You may rely on the Indyco `www.indyco.com` tool for drawing the fact schema(s). Motivate your design decisions and discuss dinamicity in dimensions.

3 Data warehouse ROLAP logical design

Starting from the conceptual design above and possibly refining the assumptions about data volumes and workload, develop a ROLAP logical design for the datawarehouse. The design must include secondary events (i.e., views). Define your fact and dimension tables in PostgreSQL and populate them with data from the operational sources. Any ETL approach is allowed, just motivate and illustrate in the documentation your choices, and include all the relevant files in the zip.

4 OLAP Queries

Specify in Postgres the queries corresponding to the workload. Moreover, referring to the specific OLAP extensions of PostgreSQL for windows and window functions, specify at least a query for each category below

- Comparison of detailed and summarized data [window partitioning]
- Computing rankings [window ordering]
- Computing cumulative totals [window framing]
- Computing mobile aggregates [window framing]

5 Hive

Import your datawarehouse (at least a relevant portion of it) in Hive and run the OLAP queries in the workload in Hive. Specify at least five new relevant queries and run such queries and the ones in the workload on Hive.

6 SparkSQL

Define the Schema RDDs corresponding to your datawarehouse (at least a relevant portion of it) and create them starting from a connection to the PostgreSQL server. Specify at least five new relevant queries as RDD operations and run them. Run the queries above and the queries in the workload on the schema RDDs.

7 Tableau

Identify the five most relevant outcomes of your analysis and provide suitable reports in Tableau for communicating this outcomes. Include in the documentation the graphics and dicuss the identified outcomes.

A Install docker and run the container

Ubuntu/Debian

You can install docker using apt: `apt-get install docker.io`.

Other Operating Systems

Go to `https://docs.docker.com/engine/installation/` and follow the instructions for your system.

Run the container

Run the container:

```
$ sudo docker run -d --name dwws --restart always \
-v /home/user/share/:/home/student/share/ acrrd/dw-workstation
```

where `/home/user/share` is the absolute path to the directory you want to use to share data with the container. The command download the image (∼1.3Gb) if not already present on the system. You have to run this command only once. You can use: `sudo docker ps -a` to see the active containers. You can use: `sudo docker start dwws` to start the container and `sudo docker stop dwws` to stop it.
The container embeds a working version of HIVE installed and it runs an ssh server. We just need to know its ip address to connect to it, the command `sudo docker inspect dwws` output information about the container, search for `"IPAddress"` to find the ip.

```
$ sudo docker inspect dwws | grep \"IPAddress\"
   "IPAddress": "172.17.0.2",
          "IPAddress": "172.17.0.2",
```

We can login on the container using ssh with user `student` and password `foobar`: `ssh student@172.17.0.2`. If the connection does not work try to start the container.

Hive. On your host machine, move the files you need to use from Hive (e.g, for importing data into Hive) in the directory you want to share with the container.
On the container, runing `ls share` you should be able to see the two files above.
If you delete the container you loose all the data except the one in `/home/user/share`.
In case you need it, root password is `toor`.
To launch the HIVE shell we just need to run the command `hive`.
For importing data, first define the table schema with

```
hive> CREATE TABLE MyTable (attr1 type1, attr2 type2, ..., attrn typen)
      row format delimited fields terminated by ',';
```

Next, we can just use the following code to load the data from the `MyFile.csv` file into the table we just created.

```
hive> LOAD DATA LOCAL INPATH '/home/student/share/MyFile.csv'
      OVERWRITE INTO TABLE MyTable;
```

Spark SQL. We can create a Spark SQL Data Frame from the csv file `MyFile.csv` by relying on the `csv` method of the `DataFrameReader` class (the use is analogous to `json` for loading from JSON):

```
.csv("/home/student/share/MyFile.csv")
```

We can make use of the `schemaInfer` option to infer the schema of the DataFrame.
E.g.:

```
Dataset<Row> csvDF = spark.read()
              .format("csv")
              .option("header", "true")
              .option("schemaInfer","true")
              .option("nullValue", "")
              .csv("/home/student/share/MyFile.csv")
              ;
```

We can then make use of `show`, `printSchema` and `count` DataFrame operations.

We can create a database with

```
$ createdb -U student dbname
```

and open the PostgreSQL client with

```
$ psql -U student dbname
```

We can access postgresql as user `'student'` password `'foobar'`.
Program `SparkSql.java` contains a small example that loads a table from PostgreSQL. We can compile it as

```
javac -cp \
 /opt/spark-2.0.1-bin-hadoop2.7/jars/spark-core_2.11-2.0.1.jar: \
 /opt/spark-2.0.1-bin-hadoop2.7/jars/spark-sql_2.11-2.0.1.jar: \
 /opt/spark-2.0.1-bin-hadoop2.7/jars/spark-hive_2.11-2.0.1.jar: \
 /opt/spark-2.0.1-bin-hadoop2.7/jars/scala-library-2.11.8.jar: \
 /opt/spark-2.0.1-bin-hadoop2.7/jars/spark-catalyst_2.11-2.0.1.jar
sparksql/SparkSql.java

jar cf sparksql.jar sparksql/
```

and execute it as

```
spark-submit \
--driver-class-path /home/student/postgresql-9.4.1212.jre6.jar \
--jars /home/student/postgresql-9.4.1212.jre6.jar \
--class sparksql.SparkSql.java sparksql.jar
```