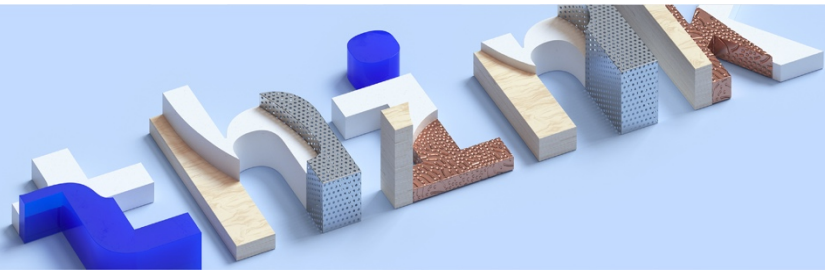


think 2018



Lab Center – Hands-on Lab

Session 4636

Session Title

Integrating Relational Data with Hadoop and Spark

Stefan Hummel, IBM, stefan.hummel@de.ibm.com

Andreas Weininger, IBM, andreas.weininger@de.ibm.com

Table of Contents

Disclaimer.....	3
Objectives of this lab.....	6
Background of this Lab.....	7
How to use the VMware Image.....	8
Preparation:	9
Lab 1: Copying files into HDFS.....	11
Lab 2: Create schema corresponding to the files copied to HDFS.....	13
Lab 3: Creating Tables in Parquet Format	18
Lab 4: Running queries with Big SQL	22
Lab 5: Running queries with Hive	24
Lab 6: Analyzing Query Plans	25
Lab 7: Linear Regression in SQL	27
Lab 8 Linear Regression with Python.....	35
Lab 9: Using Scala, Spark and Big SQL	38
Lab 10: Using Jupyter Notebooks, Spark and Db2 Warehouse	43
We Value Your Feedback!	48

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.

In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com and [names of other referenced IBM products and services used in the presentation] are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

© 2018 International Business Machines Corporation. No part of this document may be reproduced or transmitted in any form without written permission from IBM.

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Objectives of this lab

The goal of this lab is to show how data in Hadoop system can be analyzed and integrated with relational data.

Attendees will learn:

- How data can be loaded into HDFS
- How tables can be created with Big SQL and shared with Hive
- How Hive/Big SQL tables can be loaded
- How queries can be executed with different tools, and what the implication of these tools are
- How to analyze queries
- How can Python be used with Big SQL
- How Spark jobs can be integrated with Big SQL
- How Spark jobs can be integrated with Db2 Warehouse

Scripts with solutions for all of the lab exercises are provided, but it is recommended that you first try to create a script on your own before using the lab script.

Background of this Lab

This lab uses two datasets.

The first part of the lab uses the schema, the load files and the queries of the Star Schema Benchmark defined by Pat O'Neil (<http://www.cs.umb.edu/~poneil/StarSchemaB.PDF>). You find a copy of it on your desktop in the VMware image.

The Star Schema Benchmark was chosen since it simulates a typical star schema data mart, and provides all the required information for our lab (schema, data, queries).

The files for the Star Schema Benchmark are in the VMware image in the directory
~user1/Lab/SSB.

The next part of the lab uses real data from the U.S. Geological Survey (<http://www.usgs.gov/>). The data used are measurements of water resources over time. There are many different kinds of measurement, but the following labs focus on the water discharge at a specific site. Site IDs are used instead of the full site names.

How to use the VMware Image

There is a user account created for user. The password for this account is

User	Password
user	user

Commands which require root permission can be executed with the `sudo` or the `.do` command.

The browser Firefox is installed in the VMware Image.

All exercises will be executed from the VMware Image. Therefore, the very first step is to start the VMware image and log in as user “user”.

The lab uses a Hadoop cluster in the Technical Exploration Center (TEC) in Ehningen. All exercises will be executed on this cluster. Therefore, you have to connect to the TEC. After you are logged in the VMware as user “user”, you have to open a VPN connection to the TEC. For that you need your TEC user id and password. Each participant has received this separately.

Open a terminal window.

Enter the command

```
user@linux:~$ .do openconnect ehn163.spc.ibm.com
POST https://ehn163.spc.ibm.com/
Attempting to connect to server 195.75.108.163:443
SSL negotiation with ehn163.spc.ibm.com
Connected to HTTPS on ehn163.spc.ibm.com
XML POST enabled
Please enter your username and password.
Username:<TEC User Id>
Password:<TEC Password>
POST https://ehn163.spc.ibm.com/
Got CONNECT response: HTTP/1.1 200 OK
CSTP connected. DPD 30, Keepalive 20
Connected tun0 as 172.17.8.176, using SSL
```


Now you have a VPN connect. Keep this terminal window open (you can iconize it). Open new terminal windows for the following steps.

Preparation:

All users are working on the same Big SQL cluster. The user id is `bigsql` and the password is `passw0rd`. For avoiding conflicts each lab participant is assigned a group number between 1 and 20. Please ask what your group number is.

The working directory of a group is `/home/bigsql/Lab/%GN%` where `%GN%` has to be replaced by the two digit group number e.g. for group 3 the working directory will be `/home/bigsql/Lab/03`

Whenever `%GN%` is mentioned in the rest of the document, this has to be replaced with your two digit group number.

Also all objects which would produce naming conflicts like table names or file names in HDFS will have to include the group number to make them unique. Details will be explained for the individual labs.

Solutions for exercises can be found in the directory of group 00.

The cluster consists of five nodes:

ip address	node name	purpose
172.17.64.220	bivm01.de.ibm.com	Big SQL master
172.17.64.221	bivm02.de.ibm.com	Hbase master
172.17.64.209	bivm03.de.ibm.com	Worker
172.17.64.210	bivm04.de.ibm.com	Worker
172.17.64.211	bivm05.de.ibm.com	Worker

You should login to the Big SQL master node and will also be working mostly from this node.

Open a new terminal window and enter:

```
user@linux:~$ ssh bigsql@172.17.64.220
```

```
The authenticity of host '172.17.64.220 (172.17.64.220)' can't be established.
```

```
ECDSA key fingerprint is SHA256:cqSjXqNCGBTRKcuih6JPpP+7nwgIclpI6LQF6uPkPNs.
```

```
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '172.17.64.220' (ECDSA) to the list of known hosts.
```

bigsql@172.17.64.220's password:

Enter the password passw0rd and press return.

Change the directory to Lab/%GN%:

```
[bigsql@bivm01 ~]$ cd Lab/%GN%
```

Whenever you need additional windows, just open them in this way.

Lab 1: Copying files into HDFS

1. HDFS is not a standard POSIX file system. Therefore, the Linux/Unix standard command line tools can't be used for accessing and manipulating files in HDFS. Instead the command

```
hadoop fs
```

is used for accessing and manipulating files. This command is documented at

```
https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-common/FileSystemShell.html
```

2. The directory /home/bigsql/Lab/SSB/loadfiles-1GB contains five loadfiles for the five tables of the Star Schema Benchmark (SSB):

```
/home/bigsql/Lab/SSB/loadfiles-1GB/customer.tbl  
/home/bigsql/Lab/SSB/loadfiles-1GB/lineorder.tbl  
/home/bigsql/Lab/SSB/loadfiles-1GB/supplier.tbl  
/home/bigsql/Lab/SSB/loadfiles-1GB/dates.tbl  
/home/bigsql/Lab/SSB/loadfiles-1GB/part.tbl
```

Copy these files into the files

/apps/hive/warehouse/%GN%/%TABLENAME%/%TABLENAME% in HDFS.

%GN% should be replaced with your two digit group number, and %TABLENAME% is the name of a particular table e.g. if you are group 3 and want to copy the load file for table "supplier", the target file name would be

/apps/hive/warehouse/03/supplier/supplier

First the directory /apps/hive/warehouse/%GN%/%TABLENAME% has to be created. The command for doing this is

```
hadoop fs -mkdir -p <directory name>
```

e.g. for group 3 and table supplier this would be

```
hadoop fs -mkdir -p /apps/hive/warehouse/03/supplier
```

Execute this for all tables.

Next copy the files into these new directories. This can be achieved with the command

```
hadoop fs -copyFromLocal <source file> <target file>
```

Measure how much time each copy operation takes. E.g. for group 3 and table supplier the command would be

```
time hadoop fs -copyFromLocal \  
    /home/bigsql/Lab/SSB/loadfiles-1GB/supplier.tbl \  
    /apps/hive/warehouse/03/supplier/supplier
```

3. After all files have been copied, check with the

```
hadoop fs -ls <file or directory>
```

command, whether the copying was successful.

Lab 2: Create schema corresponding to the files copied to HDFS

Now we would like to use these files as tables. A generic schema for the SSB tables can be found in `/home/bigsql/Lab/SSB/ssb-dbgen-master/ssb.ddl`.

- Next, we will create table schema definitions in Big SQL which allow to access the files copied in lab 1 to HDFS as tables in Big SQL and Hive. Copy the file `/home/bigsql/Lab/SSB/ssb-dbgen-master/ssb.ddl` to your local directory and call it `ssb%GN%-row-format-delimited.ddl`.
- Edit this file with vi.
- Rename all tables `%TABLENAME%` to `%TABLENAME%%GN%_rf` e.g. for group 3 and table `supplier` call the table `supplier03_rf`
- `CREATE EXTERNAL HADOOP TABLE` creates the meta data for a table without creating a new table. `CREATE HADOOP TABLE` also initializes an empty table. Therefore, all “`CREATE TABLE`” have to be replace by “`CREATE EXTERNAL HADOOP TABLE`” in our file. Perform this step next.
- All primary key constraints can be kept. They will not be enforced but help the optimizer to get better query plans
- Check our files in what format they are.
- The files are in pipe-separated format. Therefore, insert the clause `ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'` after the column specification of each table
- The location clause specifies the location of the file in HDFS. Therefore, insert after the format clause a location clause of the form `/apps/hive/warehouse/%GN%/TABLENAME%` for each table.
- You are now finished with editing. The create table statement for table `supplier` und group 3 should look like this:

```
CREATE EXTERNAL HADOOP TABLE SUPPLIER03_RF (  
    S_SUPPKEY      INTEGER NOT NULL PRIMARY KEY,  
    S_NAME         VARCHAR(25) NOT NULL,  
    S_ADDRESS      VARCHAR(25) NOT NULL,  
    S_CITY         VARCHAR(10) NOT NULL,  
    S_NATION       VARCHAR(15) NOT NULL,  
    S_REGION       VARCHAR(12) NOT NULL,  
    S_PHONE        VARCHAR(15) NOT NULL)  
  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'  
LOCATION '/apps/hive/warehouse/03/supplier';
```

A complete solution for group 0 can be found in `/home/bigsql/Lab/00/SSB/lab02/ssb00-row-format-delimited.ddl`

- There are two tools which you can use for executing SQL against Big SQL:

db2 - the Db2 Command Line Processor (CLP)

jsqsh - the Java SQL Shell

14. Both are part of Big SQL. For most tasks both can be used. We will first use the db2 CLP to create the tables. The Big SQL database is called BIGSQL. Therefore, connect to this database first:

```
[bigsql@bivm01 lab02]$ db2 connect to bigsql
```

Database Connection Information

```
Database server          = DB2/LINUX8664 11.1.0
SQL authorization ID     = BIGSQL
Local database alias     = BIGSQL
```

15. Next execute the create table script which we just created

```
[bigsql@bivm01 lab02]$ db2 -tvf ssb00-row-format-delimited.ddl
```

After each table a line

DB20000I The SQL command completed successfully.
should indicate a successful execution.

16. Now let us check the number of rows in each table. First we will use the db2 CLP. Execute the following commands for all tables:

```
time db2 "select count(*) from %TABLENAME%%GN%_rf"
```

The quotes around the SQL statement are necessary since the SQL statement contains characters which are interpreted by the shell. The expected number of rows is given in the table below:

Tablename	Rows
supplier	2,000
part	200,000
customer	30,000
dates	2,556
lineorder	6,001,171

The file /home/bigsql/Lab/00/SSB/lab02/count.sql contains count statements for group 0.

17. Next, let's do the same check with jsqsh. Start jsqsh:

```
[bigsql@bivm01 lab02]$ jsqsh
```

Automatically connecting with connection "bigsql". Run with --setup to disable autoconnect if necessary.

Welcome to JSqsh 4.8

Type "\help" for help topics. Using JLine.

[bivm01.de.ibm.com][bigsql] 1>

Note that you don't have to connect to BIGSQL explicitly since this is already preconfigured.

Now execute again all count statements. Note that each SQL statement has to be terminated by a semicolon ";":

e.g. for table lineorder and group 03:

```
[bivm01.de.ibm.com][bigsql] 1> select count(*) from lineorder03_rf;
```

```
+-----+
```

```
|      1 |
```

```
+-----+
```

```
| 6001171 |
```

```
+-----+
```

1 row in results(first row: 1.010s; total: 1.012s)

Do the runtimes differ significantly from the times with the db2 CLP?

Exit jsqsh by typing quit.

18. It is not only possible to access these tables with Big SQL, but also with Hive since the meta data are shared. Start hive by typing hive:

```
[bigsql@bivm01 lab02]$ hive
```

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/iop/4.2.0.0/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/iop/4.2.0.0/spark/lib/spark-assembly-1.6.1-IBM_4-hadoop2.7.2-IBM-

12.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

WARNING: Use "yarn jar" to launch YARN applications.

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/iop/4.2.0.0/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/iop/4.2.0.0/spark/lib/spark-assembly-1.6.1-IBM_4-hadoop2.7.2-IBM-12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

Logging initialized using configuration in
file:/etc/hive/4.2.0.0/0/hive-log4j.properties

Again there is no need to explicitly connect. Now let's try to run the count statements. Here is an example for group 0 and table lineorder:

```
hive> select count(*) from lineorder00_rf;  
FAILED: SemanticException [Error 10001]: Line 1:21 Table not found 'lineorder00_rf'
```

To avoid this error the schema bigsql has to be specified:

```
hive> select count(*) from bigsql.lineorder00_rf;  
Query ID = bigsql_20170916093753_95f5813b-92f4-4ea3-b05e-958914782c3b  
Total jobs = 1  
Launching Job 1 out of 1  
Number of reduce tasks determined at compile time: 1  
In order to change the average load for a reducer (in bytes):  
  set hive.exec.reducers.bytes.per.reducer=<number>  
In order to limit the maximum number of reducers:  
  set hive.exec.reducers.max=<number>  
In order to set a constant number of reducers:  
  set mapreduce.job.reduces=<number>  
Starting Job = job_1495014818994_0073, Tracking URL =  
http://bivm02.de.ibm.com:8088/proxy/application_1495014818994_0073/  
Kill Command = /usr/iop/4.2.0.0/hadoop/bin/hadoop job -kill job_1495014818994_0073  
Hadoop job information for Stage-1: number of mappers: 3; number of reducers: 1  
2017-09-16 09:38:08,877 Stage-1 map = 0%, reduce = 0%  
2017-09-16 09:38:17,446 Stage-1 map = 33%, reduce = 0%, Cumulative CPU 5.21 sec  
2017-09-16 09:38:19,802 Stage-1 map = 67%, reduce = 0%, Cumulative CPU 10.24 sec  
2017-09-16 09:38:20,860 Stage-1 map = 78%, reduce = 0%, Cumulative CPU 15.99 sec  
2017-09-16 09:38:21,949 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 16.87 sec  
2017-09-16 09:38:23,043 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 18.79 sec  
MapReduce Total cumulative CPU time: 18 seconds 790 msec  
Ended Job = job_1495014818994_0073  
MapReduce Jobs Launched:  
Stage-Stage-1: Map: 3 Reduce: 1 Cumulative CPU: 18.79 sec HDFS Read: 588840973 HDFS  
Write: 8 SUCCESS  
Total MapReduce CPU Time Spent: 18 seconds 790 msec  
OK  
6001171  
Time taken: 32.229 seconds, Fetched: 1 row(s)
```


We notice that Hive gives a lot of additional information about the execution by default. Hive creates Map/Reduce jobs for executing queries. We get information about the individual stages.

Compare the runtime of the count statement with Hive to the runtime with Big SQL. Perform this for all 5 SSB tables. What do you notice?

Lab 3: Creating Tables in Parquet Format

19. In this lab we will create tables in a format which allows faster queries in many cases than the pipe delimited format which we have used up to now. The format we will use is Parquet (<http://parquet.apache.org/documentation/latest/>) which is some kind of columnar format.
20. First, we will create table schema definitions for the 5 SSB tables in Big SQL. The easiest way is probably by modifying the file `ssb%GN%-row-format-delimited.ddl` which you created in lab 2. Create a copy of this file with `cp` which should be named `ssb%GN%-parquet.ddl`
21. Edit this file with `vi`.
22. Rename all tables `%TABLENAME%%GN%_rf` to `%TABLENAME%%GN%_pq` e.g. for group 3 and table supplier call the table `supplier03_pq`.
23. Since we want to create new tables instead of reusing existing files, the `CREATE EXTERNAL HADOOP TABLE` has to be replaced by `CREATE HADOOP TABLE` for all tables. Perform this step next.
24. For getting the Parquet storage format “`ROW FORMAT DELIMITED FIELDS TERMINATED BY ' | '`” after the column specification of each table has to be replaced by “`STORED AS PARQUETFILE`”
25. Finally append `pq` to the location in the `LOCATION` clause i.e. change this to “`LOCATION '/apps/hive/warehouse/%GN%/%TABLENAME%_pq'`” for each table.
26. You are now finished with editing. The create table statement for table supplier und group 3 should look like this:

```
CREATE EXTERNAL HADOOP TABLE SUPPLIER00_PQ (  
    S_SUPPKEY      INTEGER NOT NULL PRIMARY KEY,  
    S_NAME         VARCHAR(25) NOT NULL,  
    S_ADDRESS      VARCHAR(25) NOT NULL,  
    S_CITY         VARCHAR(10) NOT NULL,  
    S_NATION       VARCHAR(15) NOT NULL,  
    S_REGION       VARCHAR(12) NOT NULL,  
    S_PHONE        VARCHAR(15) NOT NULL)  
  
STORED AS PARQUETFILE  
LOCATION '/apps/hive/warehouse/00/supplier_pq';
```

A complete solution for group 0 can be found in
/home/bigsql/Lab/00/SSB/lab03/ssb00-parquet.ddl

27. Make sure that you are still connected to BIGSQL. Then execute the file which you just created with db2 CLP. E.g. for group 3:

```
[bigsql@bivm01 lab03]$ time db2 -tvf ssb03-parquet.ddl
```

Check that you get the message

```
DB20000I  The SQL command completed successfully.
```

for each table.

28. Check the number of rows for the `lineorder` table with one of the methods discussed in lab 2. 0 rows is the expected result since we haven't loaded any rows yet.

29. Now we will populate the new `_pq` tables via insert and select from the old `_rf` tables. For this create a file `ssb%GN%-insert.sql`. It should contain an insert statement of the following kind for each table:

```
insert into bigsql.%TABLENAME%%GN%_pq
select * from bigsql.%TABLENAME%%GN%_rf;
```

A complete file for group 0 can be found at /home/bigsql/Lab/00/SSB/lab03/ssb00-insert.sql

30. The `-i` option of `jsqsh` allows to specify a file with SQL statements to be executed. We will use this to execute the file which we just created. E.g. for group 3:

```
[bigsql@bivm01 lab03]$ time jsqsh -i ssb00-insert.sql
```

Automatically connecting with connection "bigsql". Run with `--setup` to disable autoconnect if necessary.

```
30000 rows affected (total: 1.643s)
200000 rows affected (total: 1.438s)
2556 rows affected (total: 2.621s)
2000 rows affected (total: 0.584s)
6001171 rows affected (total: 5.680s)
```

```
real    0m12.884s
user    0m1.765s
sys     0m0.143s
```

31. Use the methods discussed in steps 16 to 18 to count the rows in the `_pq` tables.

32. Another option of getting the data into the new tables is loading them from files in HDFS.

33. For this create a new file `ssb%GN%-load.sql` and edit it.

34. Write a load statement for each table of the following form in that file:

```
load hadoop using
  file url '/apps/hive/warehouse/%GN%/TABLENAME%/TABLENAME%'
  with SOURCE PROPERTIES ('field.delimiter'='|')
  into table bigsql.%TABLENAME%%GN%_pq overwrite;
```

The overwrite clause is necessary since we have already inserted data in the tables. A complete file for group 0 can be found at `/home/bigsql/Lab/00/SSB/lab03/ssb00-load.sql`

35. Next execute this file again with `jsqsh` e.g. for group 0:

```
[bigsql@bivm01 lab03]$ time jsqsh -i ssb00-load.sql
Automatically connecting with connection "bigsql". Run with --setup to disable autoconnect if
necessary.
WARN [State: 0 ][Code: 5108]: The LOAD HADOOP statement completed. Number of rows loaded
into the Hadoop table: "30000". Total number of source records: "30000". If the source is
a file, number of lines skipped: "0". Number of source records that were rejected: "0". Job
identifier: "job_1495014818994_0074".. SQLCODE=5108, SQLSTATE=0 , DRIVER=3.71.24
0 rows affected (total: 36.605s)
WARN [State: 0 ][Code: 5108]: The LOAD HADOOP statement completed. Number of rows loaded
into the Hadoop table: "200000". Total number of source records: "200000". If the source
is a file, number of lines skipped: "0". Number of source records that were rejected: "0".
Job identifier: "job_1495014818994_0075".. SQLCODE=5108, SQLSTATE=0 , DRIVER=3.71.24
0 rows affected (total: 29.527s)
WARN [State: 0 ][Code: 5108]: The LOAD HADOOP statement completed. Number of rows loaded
into the Hadoop table: "2556". Total number of source records: "2556". If the source is a
file, number of lines skipped: "0". Number of source records that were rejected: "0". Job
identifier: "job_1495014818994_0076".. SQLCODE=5108, SQLSTATE=0 , DRIVER=3.71.24
0 rows affected (total: 26.128s)
WARN [State: 0 ][Code: 5108]: The LOAD HADOOP statement completed. Number of rows loaded
into the Hadoop table: "2000". Total number of source records: "2000". If the source is a
file, number of lines skipped: "0". Number of source records that were rejected: "0". Job
identifier: "job_1495014818994_0077".. SQLCODE=5108, SQLSTATE=0 , DRIVER=3.71.24
0 rows affected (total: 26.512s)
WARN [State: 0 ][Code: 5108]: The LOAD HADOOP statement completed. Number of rows loaded
into the Hadoop table: "6001171". Total number of source records: "6001171". If the source
is a file, number of lines skipped: "0". Number of source records that were rejected: "0".
Job identifier: "job_1495014818994_0078".. SQLCODE=5108, SQLSTATE=0 , DRIVER=3.71.24
0 rows affected (total: 1m5.762s)

real    3m5.456s
user    0m1.901s
sys 0m0.140s
```

What did you notice?

36. Check again the number of rows in the `_pq` tables. A script for group 0 can be found at `/home/bigsql/Lab/00/SSB/lab03/count.sql`

Lab 4: Running queries with Big SQL

37. In this lab we will run the SSB queries against the tables which we created in the previous labs.
38. There are templates for these queries for the `_rf` tables in `/home/bigsql/Lab/SSB/queries_rf`. `%GN%` in these queries must be replaced with your own two digit group number. This can be done by first creating a subdirectory `queries_rf` in your working directory:

```
mkdir queries_rf
```

39. Switch to this directory

```
cd queries_rf
```

40. The creation of your query files can be done with this for loop which substitutes `%GN%` with the group number (You have to replace the red marked `00` with your own double digit group number):

```
for f in /home/bigsql/Lab/SSB/queries_rf/q*sql
do
    fb=`basename $f`
    sed 's/%GN%/00/g' < $f > $fb
done
```

41. Run all queries with `jsqsh`:

```
cat q*sql | time jsqsh
```

The output for a query should look like this:

```
[bivm01.de.ibm.com][bigsql] 1>
[bivm01.de.ibm.com][bigsql] 2> select sum(lo_extendedprice*lo_discount) as revenue
[bivm01.de.ibm.com][bigsql] 3> from BIGSQL.lineorder00_rf, BIGSQL.dates00_rf
[bivm01.de.ibm.com][bigsql] 4> where lo_orderdate = d_datekey
[bivm01.de.ibm.com][bigsql] 5> and d_yearmonthnum = 199401
[bivm01.de.ibm.com][bigsql] 6> and lo_discount between 4 and 6
[bivm01.de.ibm.com][bigsql] 7> and lo_quantity between 26 and 35;
+-----+
|      REVENUE      |
+-----+
| 98314553869      |
+-----+
1 row in results(first row: 1.460s; total: 1.462s)
```

Capture the runtimes.

42. Next repeat this with the Parquet tables by first creating a subdirectory `queries_pq` in your working directory:

```
mkdir queries_pq
```

43. Switch to this directory

```
cd queries_rpq
```

44. The creation of your query files can be done with this for loop which substitutes `%GN%` with the group number (You have to replace the red marked `00` with your own double digit group number):

```
for f in /home/bigsql/Lab/SSB/queries_pq/q*sql
do
    fb=`basename $f`
    sed 's/%GN%/00/g' < $f > $fb
done
```

45. Run all queries with `jsqsh`:

```
cat q*sql | time jsqsh
```

Capture the run times.

46. Create a new version of query `q4_2.sql` where for `lineorder` table the `_rf` and for the remaining tables the `_pq` version is used. Execute this query with `jsqsh`

Lab 5: Running queries with Hive

47. In this lab we will rerun the queries which we wrote in lab 4 with Hive.

48. In a directory containing the sub directories `query_rf` and `query_pq` with the corresponding SQL statements for the queries make log file directories `logs_rf` and `logs_pq`:

```
mkdir logs_rf
mkdir logs_pq
```

49. Run in a for loop all `q1.sql`, `q2.sql`, and `q3.sql` queries in the `_rf` version, and write the results and logs in the log directory `logs_rf`:

```
for q in queries_rf/q[123].sql
do
    b=`basename $q .sql`
    (hive -f $q 2>&1) | tee logs_rf/$b.log
done
```

50. Run in a for loop all `q1.sql`, `q2.sql`, and `q3.sql` queries in the `_pq` version, and write the results and logs in the log directory `logs_pq`:

```
for q in queries_pq/q[123].sql
do
    b=`basename $q .sql`
    (hive -f $q 2>&1) | tee logs_pq/$b.log
done
```

51. How do the times compare to the times with Big SQL in lab 4?

Lab 6: Analyzing Query Plans

52. In this lab we will have a closer look on how execution plans look like and how they can be printed. We will use query q4_2 of SSB as an example. Therefore, copy a _pq and a _rf version of these queries into your current working directory and call them q4_2-pq-explain.sql and q4_2-rf-explain.sql respectively.
53. Edit both files and insert “explain plan for” before the select keyword.
54. Make sure that the db2 CLP has still a connection to BIGSQL.
55. Run the file q4_2-pq-explain.sql with the db2 CLP. This will not execute the query itself but populate explain tables:

```
[bigsql@bivm01 lab06]$ time db2 -tvf q4_2-pq-explain.sql
explain plan for select d_year, s_nation, p_category, sum(lo_revenue -
lo_supplycost) as profit from BIGSQL.dates00_pq, BIGSQL.customer00_pq,
BIGSQL.supplier00_pq, BIGSQL.part00_pq, BIGSQL.lineorder00_pq where lo_custkey =
c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate =
d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (d_year = 1997 or
d_year = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2') group by d_year,
s_nation, p_category order by d_year, s_nation, p_category
DB20000I The SQL command completed successfully.
```

```
real      0m0.374s
user      0m0.024s
sys 0m0.018s
```

56. Next write the contents of the explain tables with the command db2exfmt into a file q4_2_pq.exfmt readable by humans:

```
[bigsql@bivm01 lab06]$ db2exfmt -d bigsql -1 -o q4_2-pq.exfmt
DB2 Universal Database Version 11.1, 5622-044 (c) Copyright IBM
Corp. 1991, 2015
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool
```

```
Connecting to the Database.
Connect to Database Successful.
Output is in q4_2-pq.exfmt.
Executing Connect Reset -- Connect Reset was Successful.
```

57. Repeat the last two steps also for the file q4_2-rf-explain.sql:

```
[bigsql@bivm01 lab06]$ time db2 -tvf q4_2-rf-explain.sql
explain plan for select d_year, s_nation, p_category, sum(lo_revenue -
lo_supplycost) as profit from BIGSQL.dates00_rf, BIGSQL.customer00_rf,
BIGSQL.supplier00_rf, BIGSQL.part00_rf, BIGSQL.lineorder00_rf where lo_custkey =
c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate =
d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (d_year = 1997 or
d_year = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2') group by d_year,
s_nation, p_category order by d_year, s_nation, p_category
DB20000I The SQL command completed successfully.
```

```
real      0m0.385s
user      0m0.021s
sys 0m0.018s
```

```
[bigsql@bivm01 lab06]$ db2exfmt -d bigsql -1 -o q4_2-rf.exfmt
DB2 Universal Database Version 11.1, 5622-044 (c) Copyright IBM Corp. 1991, 2015
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool
```

```
Connecting to the Database.
Connect to Database Successful.
Output is in q4_2-rf.exfmt.
```

```
Executing Connect Reset -- Connect Reset was Successful.
```

58. Look at the query plans which you created. If you are familiar with DB2 query plans, what differences do you notice? What differences do you notice between the query plan for the _rf and the _pq tables?

59. Now let's look at Hive plans. For that make copies of q4_2-pq-explain.sql and q4_2-rf-explain.sql called q4_2-pq-explain-hive.sql and q4_2-rf-explain-hive.sql.

60. In these files replace "explain plan for" with "explain extended". Now the files are ready for use with Hive.

61. Generate explain files from these SQL files with Hive:

```
hive -f q4_2-pq-explain-hive.sql > q4_2-pq.explain-hive
hive -f q4_2-rf-explain-hive.sql > q4_2-rf.explain-hive
```

62. What can be seen from the files q4_2-pq.explain-hive and q4_2-rf.explain-hive?

Lab 7: Linear Regression in SQL

63. Next, we will look how Big SQL can be used for some advanced analytics. The example which we chose is Linear Regression. We will try to implement that directly in SQL. Big SQL offers a rich set of SQL functionality. The files necessary for this lab are located in the directory `/home/bigsql/Lab/linear-regression`. It contains a load file `lf_50999999.unl` of USGS data.

64. First, we create a the directory in HDFS for storing the load file (%GN% has to be replaced with the double digit group id):

```
hadoop fs -mkdir -p /apps/hive/warehouse/00/linearregression/t_50999999
```

65. Now we copy the load file into this directory:

```
hadoop fs -copyFromLocal /home/bigsql/Lab/linear-regression/lf_50999999.unl \  
/apps/hive/warehouse/%GN%/linearregression/t_50999999/t_50999999
```

66. Let's check the size of the file in HDFS:

```
[bigsql@bivm01 linear-regression]$ hadoop fs -ls  
/apps/hive/warehouse/00/linearregression/t_50999999/t_50999999-rw-r--r--  3 bigsql hadoop  
69142 2017-09-16 20:05 /apps/hive/warehouse/00/linearregression/t_50999999/t_50999999
```

67. The directory `/home/bigsql/Lab/linear-regression` contains also already a file with a table definition, whose content is shown below:

```
create external hadoop table t_50999999_%GN% (  
  c_agency_cd VARCHAR(5),  
  c_site_no VARCHAR(15),  
  c_datetime TIMESTAMP,  
  c_tz_cd VARCHAR(6),  
  c_03_00045 FLOAT,  
  c_03_00045_cd VARCHAR(10),  
  c_14_00035 FLOAT,  
  c_14_00035_cd VARCHAR(10),  
  c_15_00036 FLOAT,  
  c_15_00036_cd VARCHAR(10),  
  c_17_00021 FLOAT,  
  c_17_00021_cd VARCHAR(10),  
  c_18_00052 FLOAT,  
  c_18_00052_cd VARCHAR(10),
```

```

c_19_00025 FLOAT,
c_19_00025_cd VARCHAR(10),
c_26_00045 FLOAT,
c_26_00045_cd VARCHAR(10),
c_28_62608 FLOAT,
c_28_62608_cd VARCHAR(10)
) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
LOCATION '/apps/hive/warehouse/%GN%/linearregression/t_50999999';

```

68. %GN% must be replaced with your group id. This can be done when you are in your local working directory of your group with the following command (the red 00 must be replaced with your double digit group id):

```

sed 's/%GN%/00/g' /home/bigsql/Lab/linear-regression/create-table-50999999.sql \
> create-table-50999999.sql

```

69. Create the table by executing the script with jsqsh:

```

[bigsql@bivm01 lab07]$ time jsqsh -i create-table-50999999.sql
Automatically connecting with connection "bigsql". Run with --setup to disable
autoconnect if necessary.
0 rows affected (total: 5.300s)

real    0m6.278s
user    0m1.849s
sys 0m0.140s

```

70. Check the size of the table.

71. At next we want to find out whether any columns in our data set are correlated. We will try do this with standard SQL. We will use the Pearson's correlation coefficient (https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient) for determining whether two columns are correlated. The Pearson's correlation coefficient is defined in the following way:

$$\text{cov}(X,Y) / (\sigma_X \sigma_Y)$$

72. Big SQL contains built-in functions covariance and stddev. Write a SQL statement to check the correlation of columns C_19_00025 and C_17_00021 of table T_509999999_%GN% and execute it with jsqsh.

73. There is already a script /home/bigsql/Lab/linear-regression/correlation-usgs.sql for checking the correlation of all columns. In this script %GN% has to be

replaced by your double digit group id. This can be done with the following command, if you are in your local working directory (the red 00 must be replaced with your double digit group id):

```
sed 's/%GN%/00/g' /home/bigsql/Lab/linear-regression/correlation-usgs.sql \
> correlation-usgs.sql
```

74. Now execute the script with jsqsh:

```
[bigsql@bivm01 lab07]$ time jsqsh -i correlation-usgs.sql
Automatically connecting with connection "bigsql". Run with --setup
to disable autoconnect if necessary.
```

CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR	CORR
ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT	ELAT
ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_	ION_
25_2	35_2	36_2	45_2	52_2	6260	35_2	36_2	45_2	52_2	6260	36_3	45_3	52_3	6260	45_3	52_3	6260	52_4	6260	6260
1	1	1	1	1	8_21	5	5	5	5	8_25	5	5	5	8_35	6	6	8_36	5	8_45	8_52
-.28	.695	-.49	-.11	-.90	.712	-.38	.253	.017	.170	-.02	-.54	-.03	-.58	.552	.053	.367	-.36	.134	-.03	-.67
192	77	249	129	443	03	977	52	31	02	696	584	921	290	61	43	94	505	09	910	987

1 row in results(first row: 0.383s; total: 0.395s)

```
real    0m1.236s
user    0m1.714s
sys     0m0.117s
```

75. Which columns are correlated most?

76. In the previous step we have determined that the columns C_17_00021 and C_18_00052 are correlated most. There is a file parameter_cd_query.txt in the directory /home/bigsql/Lab/linear-regression/. In this file we can look up what the semantics of the different columns are. We find that C_17_00021 gives the air temperature in F, and C_18_00052 gives the relative humidity in percent. We want to use a linear model with C_17_00021 as the dependent variable and C_18_00052 as the independent variable. We call the parameters of the model w0 and w1. Therefore, our linear function is $w_0 + w_1 * C_{18_00052}$. For the minimum of the loss function we get the values for w0 and w1 with the following formulas:

$$w_0 = \text{avg}(C_{17_00021}) - w_1 * \text{avg}(C_{18_00052})$$

$$w_1 = \frac{(\text{avg}(C_{18_00052} * C_{17_00021}) - \text{avg}(C_{18_00052}) * \text{avg}(C_{17_00021}))}{(\text{avg}(C_{18_00052} * C_{18_00052}) - \text{avg}(C_{18_00052}) * \text{avg}(C_{18_00052}))}$$

77. To be able to compute the parameters for our model on part of the data and test it on another part of the data, we partition our data into those rows which have C_DATETIME < 2014-07-21 10:00:00 which will be the data used for learning, and into those rows which have C_DATETIME >= 2014-07-21 10:00:00 which will be used for testing.

78. We will be doing this partitioning by defining two views `t_50999999_learn` and `t_50999999_test`. In the directory `/home/bigsql/Lab/linear-regression/` there is already a script `create-views.sql` which creates these two views. For using this script you have to replace `%GN%` with your double digit group id. If you are in your local working directory, this can be achieved with the following command (the red `00` must be replaced with your double digit group id):

```
sed 's/%GN%/00/g' /home/bigsql/Lab/linear-regression/create-views.sql \
> create-views.sql
```

79. Execute the script with `jsqsh`:

```
[bigsql@bivm01 lab07]$ jsqsh -i create-views.sql
Automatically connecting with connection "bigsql". Run with --setup to disable
autoconnect if necessary.
0 rows affected (total: 0.152s)
0 rows affected (total: 0.010s)
```

80. Check the number of rows corresponding to each view:

```
bigsql@bivm01 lab07]$ jsqsh
Automatically connecting with connection "bigsql". Run with --setup to disable
autoconnect if necessary.
Welcome to JSqsh 4.8
Type "\help" for help topics. Using JLine.
bivm01.de.ibm.com][bigsql] 1> select count(*) from bigsql.t_50999999_00_learn;
+-----+
| 1 |
+-----+
| 424 |
+-----+
1 row in results(first row: 0.224s; total: 0.226s)
[bivm01.de.ibm.com][bigsql] 1> select count(*) from bigsql.t_50999999_00_test;
+-----+
| 1 |
+-----+
| 308 |
+-----+
1 row in results(first row: 0.246s; total: 0.247s)
[bivm01.de.ibm.com][bigsql] 1> select count(*) from bigsql.t_50999999_00;
+-----+
| 1 |
+-----+
| 732 |
+-----+
1 row in results(first row: 0.243s; total: 0.244s)
```

81. Now check the number of rows in the views also with Hive:

```
[bigsql@bivm01 lab07]$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/iop/4.2.0.0/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/iop/4.2.0.0/spark/lib/spark-assembly-1.6.1-IBM_4-hadoop2.7.2-IBM-12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
WARNING: Use "yarn jar" to launch YARN applications.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/iop/4.2.0.0/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/iop/4.2.0.0/spark/lib/spark-assembly-1.6.1-IBM_4-hadoop2.7.2-IBM-12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

Logging initialized using configuration in file:/etc/hive/4.2.0.0/0/hive-log4j.properties
hive> select count(*) from bigsql.t_50999999_00;
Query ID = bigsql_20170916224102_0c400d68-b506-4a9c-b854-1a50247c201d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1495014818994_0174, Tracking URL
http://bivm02.de.ibm.com:8088/proxy/application_1495014818994_0174/
Kill Command = /usr/iop/4.2.0.0/hadoop/bin/hadoop job -kill job_1495014818994_0174
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2017-09-16 22:41:16,121 Stage-1 map = 0%, reduce = 0%
2017-09-16 22:41:21,493 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.34 sec
2017-09-16 22:41:26,938 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.3 sec
MapReduce Total cumulative CPU time: 4 seconds 300 msec
Ended Job = job_1495014818994_0174
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.3 sec HDFS Read: 79299 HDFS
Write: 4 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 300 msec
OK
732
Time taken: 26.08 seconds, Fetched: 1 row(s)
hive> select count(*) from bigsql.t_50999999_00_learn;
FAILED: SemanticException [Error 10001]: Line 1:21 Table not found
't_50999999_00_learn'
hive> select count(*) from bigsql.t_50999999_00_test;
```

```
FAILED: SemanticException [Error 10001]: Line 1:21 Table not found
't_50999999_00_test'
```

Note that while tables created with Big SQL are visible for Hive and vice versa, the same is not true for views.

82. Now write an SQL script which computes the parameters of this linear regression model. Use the formulas shown above. Execute the script and write down the parameters.

A solution for this script can be found in the file `get-model-parameters.sql` in the directory `/home/bigsql/Lab/linear-regression` (as usual %GN% has to be replaced with your double digit group id):

```
WITH h AS (
  SELECT
    AVG(C_17_00021) avg_21,
    AVG(C_18_00052) avg_52,
    AVG(C_17_00021*C_18_00052) avg_21_52,
    AVG(C_18_00052*C_18_00052) avg_52_52
  FROM
    t_50999999_%GN%_learn
)
SELECT
  avg_21 - ((avg_21_52 - avg_52 * avg_21) / (avg_52_52 - avg_52 *
    avg_52))*avg_52 AS w0,
  (avg_21_52 - avg_52 * avg_21) / (avg_52_52 - avg_52 * avg_52) AS w1
FROM
  h;
```

83. After successful execution, you should see a result showing the model parameters:

```
[bigsql@bivm01 lab07]$ jqsh -i get-model-parameters.sql
```

Automatically connecting with connection "bigsql". Run with `--setup` to disable autoconnect if necessary.

```
+-----+-----+
```



```

|          W0 |          W1 |
+-----+-----+
| 108.56495 | -.36305 |
+-----+-----+

1 row in results(first row: 0.415s; total: 0.430s)

```

84. Now we can use these parameters to predict the air temperature for the second part of the data set. Write an SQL statement for this and execute it.
85. The following SQL script determines not only the predicted values but also computes the loss to have a measure for the quality of the prediction (as usual %GN% has to be replaced with your double digit group id):

```

SELECT
    c_18_00052,
    c_17_00021,
    108.56495101346-0.3630508488702458*c_18_00052 AS prediction_21,
    (c_17_00021-108.56495101346042+0.3630508488702458*c_18_00052)*
    (c_17_00021-108.56495101346042+0.3630508488702458*c_18_00052) loss
FROM
    t_50999999_%GN%_test
;

```

86. The file predict.sql in /home/bigsql/Lab/linear-regression contains this script. Run the script with %GN% substituted. The output should look like this:

```

[bigsql@bivm01 lab07]$ jsqsh -i predict.sql

Automatically connecting with connection "bigsql". Run with --setup
to disable autoconnect if necessary.

```

```

+-----+-----+-----+-----+
| C_18_00052 | C_17_00021 | PREDICTION_21 | LOSS |
+-----+-----+-----+-----+
| 58.80000 | 86.50000 | 87.21756 | .51489 |
| 55.30000 | 88.20000 | 88.48824 | .08308 |
| 64.40000 | 86.70000 | 85.18448 | 2.29681 |

```

	64.10000		86.90000		85.29339		2.58119	
	66.70000		86.40000		84.34946		4.20472	
	66.10000		86.40000		84.56729		3.35883	
	65.20000		86.20000		84.89404		1.70554	
	66.00000		86.20000		84.60359		2.54851	
...								
	60.60000		86.90000		86.56407		.11285	
	61.30000		86.90000		86.30993		.34818	
+-----+-----+-----+-----+								

308 rows in results(first row: 0.300s; total: 0.370s)

Lab 8 Linear Regression with Python

87. In this lab, we will not compute the linear regression in Big SQL, but will extract the data to the client and process the data there. Also, we will not use a GUI, but instead use the command line. The programming language we will use is Python.

In a terminal window change the directory to `/home/bigsql/Lab/%GN%/linear-regression/lab08` with the `cd` command. Then type in python (“\$” is the prompt which you don't have to type; black is the output of the system, blue the text which you have to type yourself, <red> is information which you have to type yourself, but replace with the concrete information for your system) to get an interactive Python shell:

```
[bigsql@bivm01 lab07]$ python
Python 2.7.5 (default, Oct 11 2015, 17:47:16)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

88. For accessing the BIGSQL database we use the SQLAlchemy toolkit (<http://www.sqlalchemy.org>). First, we import the `create_engine` function which is used for creating the database connection:

```
>>> from sqlalchemy import create_engine
```

89. Next we use the `create_engine` function to initialize the information for the database connection.

```
>>> e =
create_engine("db2+ibm_db://<userid>:<password>@<hostname>:<portnumber>/<databasename>")
```

which means in our case:

```
>>> e = create_engine("db2+ibm_db://bigsql:passw0rd@bivm01.de.ibm.com:32051/BIGSQL")
```

90. Now we connect to the BIGSQL database:

```
>>> c = e.connect()
```

91. We will use the pandas library (<http://pandas.pydata.org/>) which provides data frames for Python. Therefore, we import the `DataFrame` function from this library:

```
>>> from pandas import DataFrame
```

92. On the connect `c` which we created above, we can use the `execute` method to execute SQL statement against our database. The result of type `ResultProxy` will be assigned to a variable `r` (`%GN%` has to be replaced with your double digit group id):

```
>>> r = c.execute("SELECT * FROM T_50999999_%GN%_LEARN")
```

93. We use the `fetchall` method on `r` to initialize a pandas data frame and assign this dataframe to a variable `df`:

```
>>> df = DataFrame(r.fetchall())
```

94. Next let us look at some methods for getting information about the content of the data frame. With the `head` method we can look at the first few rows. The first line of the output just numbers columns, while all the other output lines are numbered in the first column:

```
>>> df.head()
   0    1
0  74.7  81.3
1  73.7  81.3
2  73.2  81.1
3  72.2  81.1
4  71.6  81.0
```

With `values.shape`, we can get information on the number of rows and columns in the data frame:

```
>>> df.values.shape
(424, 2)
```

95. We can project individual columns with `iloc`:

```
>>> df.iloc[:,1]
0    81.3
1    81.3
2    81.1
3    81.1
4    81.0
5    80.8
6    80.8
7    80.6
8    80.6
9    80.6
10   80.6
...
```

96. Now we want to compute a linear regression model on that. For this we will use scikit-learn machine learning library (scikit-learn.org/). First, we import the linear model from the library, initialize it and pass it our values from the data frame:

```
>>> import sklearn.linear_model as sl
>>> linReg = sl.LinearRegression(fit_intercept=True)
>>> linReg.fit(y=df.iloc[:,1],X=df.iloc[:,0].values.reshape(-1,1))
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

97. We can print the parameter of our model with `intercept_` and `coef_`. Obviously, we are getting the same values as in the previous lab:

```
>>> print(linReg.intercept_)

108.564951013

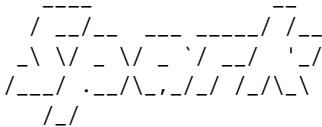
>>> print(linReg.coef_)

[-0.36305085]
```

Lab 9: Using Scala, Spark and Big SQL

98. There are two methods for integrating Spark with Big SQL. Polymorphic Table Functions allow Big SQL calling Spark code and presenting the result as a table to Big SQL. The other method is Spark using Big SQL as input for Spark computations e.g. machine learning. This lab will focus on the latter using again the data from the previous lab and linear regression as application.
99. We will use Scala as programming language for Spark. We will use the spark-shell interactively for submitting Scala code. Therefore, change to the working directory of your group and start spark-shell with the driver for Big SQL:

```
spark-shell --driver-class-path /usr/libmpacks/bigsql/4.2.0.0/db2/java/db2jcc4.jar  
17/09/16 23:48:52 WARN SparkConf: The configuration key  
'spark.yarn.applicationMaster.waitTries' has been deprecated as of Spark 1.3 and may be  
removed in the future. Please use the new key 'spark.yarn.am.waitForTime' instead.  
17/09/16 23:48:52 INFO SecurityManager: Changing view acls to: bigsql  
17/09/16 23:48:52 INFO SecurityManager: Changing modify acls to: bigsql  
17/09/16 23:48:52 INFO SecurityManager: SecurityManager: authentication disabled; ui acls  
disabled; users with view permissions: Set(bigsql); users with modify permissions:  
Set(bigsql)  
17/09/16 23:48:52 INFO HttpServer: Starting HTTP Server  
17/09/16 23:48:52 INFO Server: jetty-8.y.z-SNAPSHOT  
17/09/16 23:48:52 INFO AbstractConnector: Started SocketConnector@0.0.0.0:40923  
17/09/16 23:48:52 INFO Utils: Successfully started service 'HTTP class server' on port 40923.  
Welcome to
```



```
version 1.6.1  
  
...  
SQL context available as sqlContext.
```

scala>

100. Now let's import some libraries:

```
scala> import org.apache.spark.sql
import org.apache.spark.sql
scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row
```

101. A Spark abstraction for a relational table is the DataFrame (in Spark 2.0 there is also something additional called Dataset). Let us try to put our table from the previous example in a DataFrame (as always the red 00 has to be replaced with your double digit group id):

```
scala> val df =
sqlContext.read.format("jdbc").option("url","jdbc:db2://bivm01.de.ibm.com:32051/BIG
SQL").option("dbtable","bigsql.T_50999999_00_LEARN").option("user","bigsql").option
("password","passw0rd").load()
df: org.apache.spark.sql.DataFrame = [C_18_00052: double, C_17_00021: double]
```

102. Let us check with show whether the DataFrame contains the expected data:

```
scala> df.show()
17/09/16 23:59:24 INFO SparkContext: Starting job: show at <console>:32
17/09/16 23:59:24 INFO DAGScheduler: Got job 1 (show at <console>:32) with 1 output
partitions
17/09/16 23:59:24 INFO DAGScheduler: Final stage: ResultStage 1 (show at <console>:32)
17/09/16 23:59:24 INFO DAGScheduler: Parents of final stage: List()
17/09/16 23:59:24 INFO DAGScheduler: Missing parents: List()
17/09/16 23:59:24 INFO DAGScheduler: Submitting ResultStage 1 (MapPartitionsRDD[3] at show at
<console>:32), which has no missing parents
17/09/16 23:59:24 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated
size 4.9 KB, free 12.5 KB)
17/09/16 23:59:24 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory
(estimated size 2.4 KB, free 14.9 KB)
17/09/16 23:59:24 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on
localhost:35295 (size: 2.4 KB, free: 511.1 MB)
17/09/16 23:59:24 INFO SparkContext: Created broadcast 1 from broadcast at
DAGScheduler.scala:1006
17/09/16 23:59:24 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 1
(MapPartitionsRDD[3] at show at <console>:32)
17/09/16 23:59:24 INFO TaskSchedulerImpl: Adding task set 1.0 with 1 tasks
17/09/16 23:59:24 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, localhost,
partition 0,PROCESS_LOCAL, 1922 bytes)
17/09/16 23:59:24 INFO Executor: Running task 0.0 in stage 1.0 (TID 1)
17/09/16 23:59:24 INFO BlockManagerInfo: Removed broadcast_0_piece0 on localhost:35295 in
memory (size: 2.5 KB, free: 511.1 MB)
17/09/16 23:59:24 INFO ContextCleaner: Cleaned accumulator 1
17/09/16 23:59:25 INFO JDBCRDD: closed connection
17/09/16 23:59:25 INFO Executor: Finished task 0.0 in stage 1.0 (TID 1). 2123 bytes result
sent to driver
17/09/16 23:59:25 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 388 ms on
localhost (1/1)
17/09/16 23:59:25 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all
completed, from pool
17/09/16 23:59:25 INFO DAGScheduler: ResultStage 1 (show at <console>:32) finished in 0.391 s
17/09/16 23:59:25 INFO DAGScheduler: Job 1 finished: show at <console>:32, took 0.429201 s
+-----+
|C_18_00052|C_17_00021|
+-----+-----+
|      74.7|      81.3|
|      73.7|      81.3|
|      73.2|      81.1|
|      72.2|      81.1|
|      71.6|      81.0|
|      70.3|      80.8|
|      69.3|      80.8|
|      69.3|      80.6|
|      70.3|      80.6|
|      71.3|      80.6|
```

71.9	80.6
72.1	80.6
72.2	80.4
71.8	80.4
72.1	80.4
71.8	80.4
71.9	80.2
72.5	80.2
73.6	79.7
73.6	79.7

+-----+

only showing top 20 rows

103. Let's try to perform machine learning with this data. First import `LinearRegression` from `ml`:

```
scala> import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.regression.LinearRegression
```

104. Now instantiate the class (it is possible to set several parameter, but the details of linear regression are not the focus of this lab):

```
scala> val lr = new LinearRegression()
lr: org.apache.spark.ml.regression.LinearRegression = linReg_1efaa43d9288
```

105. The computation of the model is done with the `fit` method:

```
scala> val lrModel = lr.fit(df)
java.lang.IllegalArgumentException: Field "features" does not exist.
    at org.apache.spark.sql.types.StructType$$anonfun$apply$1.apply(StructType.scala:212)
...
```

106. Unfortunately, we get an error. This operation expects that the independent columns are called `features` and the dependent variable has to be called `label`. Let's try to fix this with a view in Big SQL. In a second window create the following view with `jsqsh`. `%GN%` has to be replaced with your double digit group id. `/home/bigsql/Lab/00/linear-regression /v_50999999.sql` contains a version for group 0. Run the view with `jsqsh`.

```
create view v_50999999_%GN%_learn (features,label) as
select c_18_00052,c_17_00021 from t_50999999_00;
```

107. Load the `DataFrame` with the view and perform the fit again (as always the red `00` has to be replaced with your double digit group id):

```
108.scala> val df = sqlContext.read.format("jdbc").option("driver",
    "com.ibm.db2.jcc.DB2Driver").option("url","jdbc:db2://bivm01.de.ibm.com:320
```



```
51/BIGSQL").option("dbtable","bigsql.V_50999999_00_LEARN").option("user","bigsql").option("password","passw0rd").load()
df: org.apache.spark.sql.DataFrame = [FEATURES: double, LABEL: double]
scala> val lrModel = lr.fit(df)
java.lang.IllegalArgumentException: Field "features" does not exist.
    at
org.apache.spark.sql.types.StructType$$anonfun$apply$1.apply(StructType.scala:212)
...
```

109. We get an error again. The column name FEATURES must be in lower case and should be a vector. Let us import some additional libraries, write a UDF to convert the column into a vector, and apply this UDF:

```
scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._

scala> import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.mllib.linalg.{Vector, Vectors}

scala> val toVec1 = udf[Vector, Double](a=>Vectors.dense(a))
toVec1: org.apache.spark.sql.UserDefinedFunction =
UserDefinedFunction(<function1>,org.apache.spark.mllib.linalg.VectorUDT@f71b0bce,List(DoubleType))

scala> val df1 = df.withColumn("features",toVec1(df("features")))
df1: org.apache.spark.sql.DataFrame = [features: vector, LABEL: double]
Let's give it a new try with the DataFrame df1:
scala> val lrModel = lr.fit(df1)
java.lang.IllegalArgumentException: Field "label" does not exist.
    at
org.apache.spark.sql.types.StructType$$anonfun$apply$1.apply(StructType.scala:212)
...
```

110. There is no longer an error with features, but the column label must also be in lower case. Let's fix that and try it again:

```
scala> val df2 = df1.withColumnRenamed("LABEL","label")
df2: org.apache.spark.sql.DataFrame = [features: vector, label: double]

scala> val lrModel = lr.fit(df2)
17/09/17 08:55:40 WARN SparkConf: The configuration key
'spark.yarn.applicationMaster.waitTries' has been deprecated as of Spark 1.3 and may be
removed in the future. Please use the new key 'spark.yarn.am.waitTime' instead.
17/09/17 08:55:40 INFO SparkContext: Starting job: first at LinearRegression.scala:163
17/09/17 08:55:40 INFO DAGScheduler: Registering RDD 110 (map at LinearRegression.scala:161)
17/09/17 08:55:40 INFO DAGScheduler: Got job 27 (first at LinearRegression.scala:163) with 1
output partitions
17/09/17 08:55:40 INFO DAGScheduler: Final stage: ResultStage 34 (first at
LinearRegression.scala:163)
```

```

17/09/17 08:55:40 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 33)

...

17/09/17 08:55:41 INFO Executor: Finished task 0.0 in stage 39.0 (TID 39). 1830 bytes result
sent to driver
17/09/17 08:55:41 INFO TaskSetManager: Finished task 0.0 in stage 39.0 (TID 39) in 10 ms on
localhost (1/1)
17/09/17 08:55:41 INFO TaskSchedulerImpl: Removed TaskSet 39.0, whose tasks have all
completed, from pool
17/09/17 08:55:41 INFO DAGScheduler: ResultStage 39 (count at LinearRegression.scala:598)
finished in 0.010 s
17/09/17 08:55:41 INFO DAGScheduler: Job 31 finished: count at LinearRegression.scala:598,
took 0.275578 s
lrModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_1efaa43d9288

scala>

```

111. Finally, it worked. Let's look at the coefficients and intercept:

```

scala> println(s"Coefficients: ${lrModel.coefficients} Intercept:
${lrModel.intercept}")
Coefficients: [-0.36883767380393134] Intercept: 109.07647739411512

```

112. If you want to know more you can get a summary and look at additional information. This should conclude our lab:

```

scala> val trainingSummary = lrModel.summary
trainingSummary: org.apache.spark.ml.regression.LinearRegressionTrainingSummary =
org.apache.spark.ml.regression.LinearRegressionTrainingSummary@1051fc4b

scala> println(s"numIterations: ${trainingSummary.totalIterations}")
numIterations: 1

```

Lab 10: Using Jupyter Notebooks, Spark and Db2 Warehouse

113. This lab will not focus on Big SQL but instead on integrating Db2 Warehouse with Spark. We will learn how to run a Spark machine learning job from a Jupyter Notebook against Db2 Warehouse. For this lab no VPN connection is necessary. Therefore, if you had started a VPN connection with openconnect for the previous labs, you can terminate it now.

114. Db2 Warehouse is already installed in your VMware machine. But you have to start it first. For this open a new terminal window and type:

```
user@linux:~$ docker start Db2wh
```

You should get the output:

```
Db2wh
```

115. You can check the status of Db2 Warehouse with

```
user@linux:~$ docker exec -it Db2wh status
```

When Db2 Warehouse has started successfully, the final output should look like:

```
Getting IBM Db2 Warehouse status...
```

```
IBM Db2 Warehouse is currently in the process of being stopped/started.  
Status may not be accurate.
```

```
-- IBM Db2 Warehouse Services Status --
```

SUMMARY

```
Db2TablesOnline      : RUNNING  
Db2connectivity      : RUNNING  
Db2running           : RUNNING  
LDAPrunning          : RUNNING  
WebConsole           : RUNNING  
Spark                : ENABLED  
LDAPsearch           : SUCCESS
```

Important:

- * If you configured authentication by using an external LDAP server, the 'LDAPrunning' status in the output should be STOPPED.
- * If the 'LDAPsearch' status in the output is SUCCESS, IBM Db2 Warehouse can access the LDAP server.

```
***** IBM Db2 Warehouse license information *****
```

```
* License type          : Trial  
* License expiry date   : 05/28/2018  
* Number of days remaining : 73  
* License status        : Active
```

```
*****
```

But it may take some minutes before this state is reached.

116. Next the container for the notebook has to be started with the following command:

```
user@linux:~$ docker run -it --rm --net=host -e DASHDBUSER=bluadmin -e DASHDBPASS=bluadmin \
dashdblocal_notebook
```

```
Using default localhost for DASHDBHOST
Verifying Spark environment on localhost
Detected build 497
```

```
Success.
```

```
Uploading /home/jovyan/resources/toree.jar to bluadmin apps directory on localhost
```

```
Upload complete:
```

```
{"result":{"filesUploaded":["spark\\apps\\toree.jar"]},"errorMessageCode":"NONE","resultCode":
:"SUCCESS","message":"NONE"}
```

```
Uploading /home/jovyan/resources/ipython-launcher.py to bluadmin apps directory on localhost
```

```
Upload complete: {"result":{"filesUploaded":["spark\\apps\\ipython-
launcher.py"]},"errorMessageCode":"NONE","resultCode":"SUCCESS","message":"NONE"}
```

```
Uploading /home/jovyan/resources/ipython-installer.py to bluadmin apps directory on localhost
```

```
Upload complete: {"result":{"filesUploaded":["spark\\apps\\ipython-
installer.py"]},"errorMessageCode":"NONE","resultCode":"SUCCESS","message":"NONE"}
```

```
Uploading /home/jovyan/resources/startup-ipython-notebook.py to bluadmin apps directory on
localhost
```

```
Upload complete: {"result":{"filesUploaded":["spark\\apps\\startup-ipython-
notebook.py"]},"errorMessageCode":"NONE","resultCode":"SUCCESS","message":"NONE"}
```

```
Checking IPython availability. This may take some time on the first attempt...
```

```
Using default localhost for DASHDBHOST
```

```
Verifying IPython installation on localhost.
```

```
Success: IPython available at /mnt/blumeta0/home/bluadmin/.local/lib/python2.7/site-
packages/ipykernel/__init__.pyc
```

```
Can use IPython kernel.
```

```
Patching /opt/conda/lib/python3.5/site-packages/notebook/templates/page.html
```

```
[I 08:56:22.878 NotebookApp] Writing notebook server cookie secret to
```

```
/home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
```

```
[W 08:56:22.983 NotebookApp] WARNING: The notebook server is listening on all IP addresses
and not using encryption. This is not recommended.
```

```
[I 08:56:23.261 NotebookApp] Loaded jupyter_cms
```

```
[I 08:56:23.273 NotebookApp] Loaded jupyter_cms_sparkapp
```

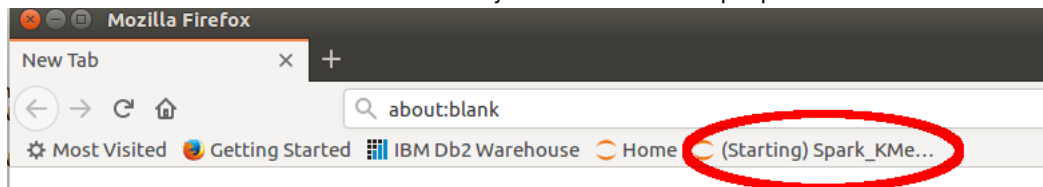
```
[I 08:56:23.284 NotebookApp] Serving notebooks from local directory: /home/jovyan/work
```

```
[I 08:56:23.284 NotebookApp] 0 active kernels
```

```
[I 08:56:23.285 NotebookApp] The Jupyter Notebook is running at: http://[all ip addresses on
your system]:8888/
```

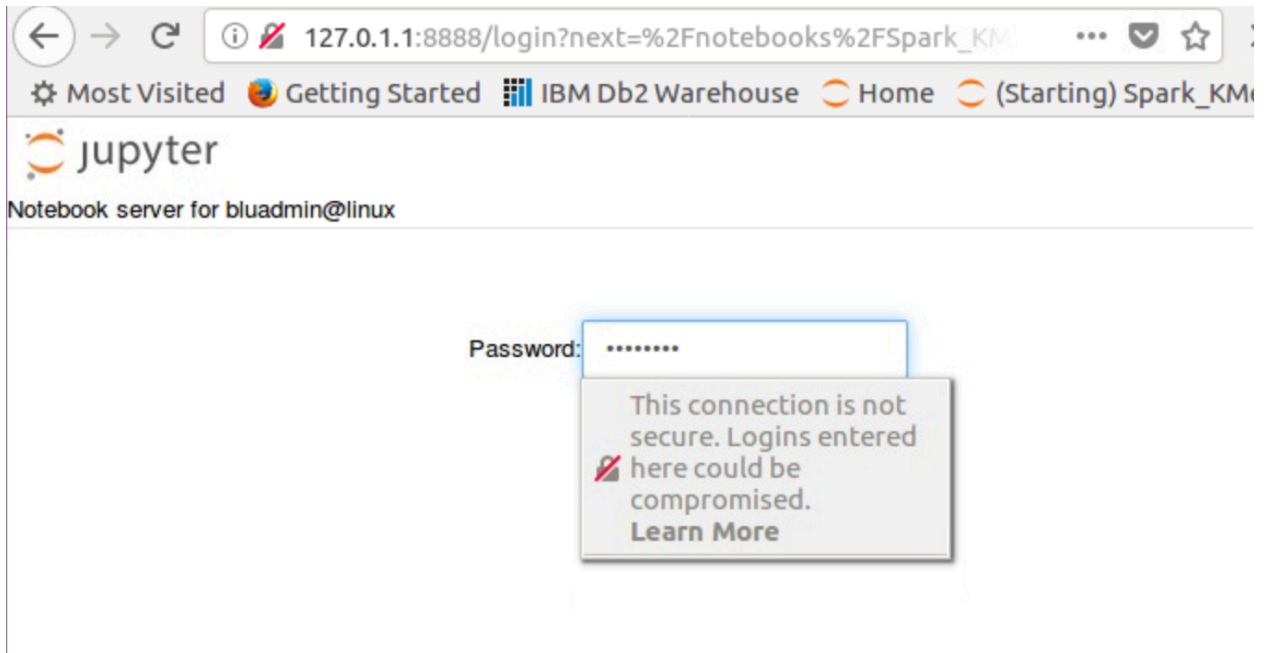
```
[I 08:56:23.285 NotebookApp] Use Control-C to stop this server and shut down all kernels
(twice to skip confirmation).
```

117. When the container is running you can start using Jupyter Notebooks (<http://jupyter.org/>). For this start the Firefox browser. There is already a bookmark for a prepared notebook:



118. Click on the bookmark in the bookmark toolbar saying “(Starting) Spark_KMe...”

119. When you get the following screen,



please enter the password [bluadmin](#)
120. Now you should see the notebook:

127.0.1.1:8888/notebooks/Spark_KMeansSample.ipynb

Most Visited Getting Started IBM Db2 Warehouse Home (Starting) Spark_KMe...

jupyter Spark_KMeansSample Last Checkpoint: 10 hours ago (autosaved)

Notebook server for bludmin@linux

File Edit View Insert Cell Kernel Help

Markdown CellToolbar

Running Spark ML on DashDB sample data

Import the necessary Spark classes

```
In [2]: import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.clustering.KMeansModel
import org.apache.spark.ml.feature.VectorAssembler
```

Load the data from the TRAINING sample. This table is pre-populated in dashDB local.

```
In [3]: val data = spark.read.format("com.ibm.idax.spark.idaxsource").
option("url", "jdbc:db2:BLUDB").
option("dbtable", "SAMPLES.TRAINING").
option("mode", "JDBC").
load()
println(data)
```

Build a Spark ML pipeline that selects the call counts from the customer data and clusters them using KMeans

```
In [9]: val assembler = new VectorAssembler().
setInputCols(Array("INTL_CALLS", "DAY_CALLS", "EVE_CALLS", "NIGHT_CALLS")).
setOutputCol("features")

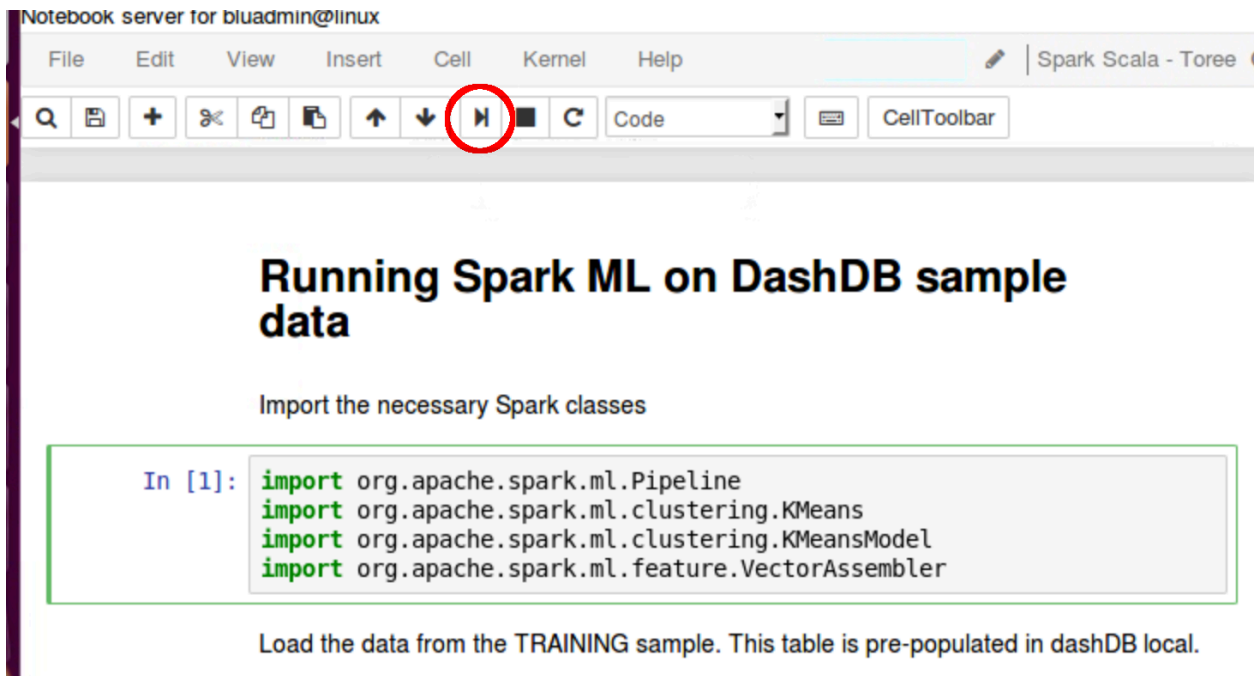
val clustering = new KMeans().
setFeaturesCol("features").
setK(3).
setMaxIter(3)

val pipe = new Pipeline().
setStages(Array(assembler, clustering))
```

Run the pipeline to find the clusters

```
In [5]: val model = pipe.fit(data)
```

121. The notebook combines documentation, code and output. Code is contained in the grey boxes. These boxes are called cells. Code in cells can be executed in several ways. One is by selecting the box and clicking on the red circled icon:



122. Perform this for the first cell. During the execution of the cell, the number in the brackets in front of the cell will be replaced by a star. When the execution is finished, the star will be replaced by a number again. Another option for executing the code in a cell is by placing the cursor in the cell and pressing <shift><return>
123. Read the documentation in the Jupyter notebook, execute the following cells one by one, look at the output and try to understand what is going on.
124. Finish the whole notebook in this way.

We Value Your Feedback!

- Don't forget to submit your Think 2018 session and speaker feedback! Your feedback is very important to us – we use it to continually improve the conference.
- Access the Think 2018 agenda tool to quickly submit your surveys from your smartphone, laptop or conference kiosk.

