Big Data – ETL Workflow

Cooperative State University Baden-Wuerttemberg

www.marcel-mittelstaedt.com

Agenda – 29.10.2018

01

Presentation and Discussion: Exercise Of Last Lecture

Spark, PySpark and Jupyter

02

HandsOn – ETL Workflow Tools (PDI)

A quick Introduction to Pentaho Data Integration

03

Exercise – Working with PDI on Hadoop and Hive

Automate the process of downloading, importing, transforming and storing IMDb data within da Hadoop Cluster.

04

Schedule

	Lecture Topic	*HandsOn*
01.10.2018 16:00-19:30 Ro. 0.11	About This Lecture, Introduction to Big Data	Hadoop
08.10.2018 16:00-19:30 Ro. 0.11	(Non-)Functional Requirements Of Distributed Data-Systems, Data Models and Access	MapReduce, Hive, HiveQL
15.10.2018 16:00-19:30 Ro. 0.11	Challenges Of Distributed Data Systems: Replication	Hive/HDFS Partitioning, HiveServer2
22.10.2018 16:00-19:30 Ro. 0.11	Challenges Of Distributed Data Systems: Partitioning	Spark, Scala and PySpark/Jupyter
29.10.2018 16:00-19:30 Ro. 0.11	**ETL Workflow And Automation**	Pentaho Data Integration
05.11.2018 16:00-19:30 Ro. 0.11	Batch and Stream Processing	MongoDB or Spark Streaming or Flink
12.11.2018 16:00-19:30 Ro. 0.11	Work On Practical Examination	
19.11.2018 16:00-19:30 Ro. 0.11	Presentation Of Practical Examination	

Solution – Exercise 04

Spark, PySpark, Jupyter

Solution

Prerequisites:

- Execute all preparation and example tasks of previous HandsOn slides of last lecture
  - Install and setup Spark
  - Install and setup PySpark
  - Install and setup Jupyter

- Start HDFS and YARN
- Start Spark/PySpark/Jupyter

See:

https://github.com/marcelmittelstaedt/BigData/tree/master/solutions/04_spark_pyspark_jupyter/JupyterExercise.html

…for complete solution (Jupyter Notebook).

Solution

Exercise 2:

a) How many **movies** are within the IMDB dataset?

```
In [9]: from pyspark.sql.functions import col
        movie_count = imdb_movie_dataframe.filter(col('titleType') == 'movie').count()

In [11]: print movie_count

499667
```

b) Who is the **oldest** actor/writer/… within the dataset?

```
In [13]: imdb_people_dataframe = spark.read.format('com.databricks.spark.csv').options(delimiter = '\t',header ='true',nullValu
         e ='null',inferSchema='true').load('hdfs:///user/hadoop/names.basics.tsv')

In [19]: oldest_one = imdb_people_dataframe.filter(col('birthYear') != '\N').sort(col("birthYear").asc()).show(3)

+---------+-----------------+---------+---------+-------------------+-------------------+
|   nconst|      primaryName|birthYear|deathYear|   primaryProfession|      knownForTitles|
+---------+-----------------+---------+---------+-------------------+-------------------+
|nm8572003|   Michael Vignola|     0001|       \N|composer,music_de...|tt6998038,tt40992...|
|nm0784172|Lucio Anneo Seneca|     0004|     0065|             writer|tt0218822,tt09725...|
|nm0194670|       Célina Cély|     0004|       \N|            actress|tt0043347,tt00560...|
+---------+-----------------+---------+---------+-------------------+-------------------+
only showing top 3 rows
```

Solution

c) Create a list (`tconst, original_title, start_year, average_rating, num_votes`) of movies which are:

- equal or newer than year 2000
- have an average rating better than 8
- have been voted more than 100.000 times

save result (DataFrame) back to HDFS as CSV File.

```
In [20]: imdb_ratings_dataframe = spark.read.format('com.databricks.spark.csv').options(delimiter = '\t',header ='true',nullVal
         ue ='null',inferSchema='true').load('hdfs:///user/hadoop/imdb_raw/title_ratings/2018/12/7/title.ratings.tsv')

In [25]: all_imdb_dataframe = imdb_ratings_dataframe.join(imdb_movie_dataframe, imdb_ratings_dataframe.tconst == imdb_movie_dat
         aframe.tconst)

In [34]: good_movies = all_imdb_dataframe.filter(col('numVotes') > 100000).filter(col('startYear') > 2000).filter(col('averageR
         ating') > 8.0).filter(col('titleType') == 'movie')

In [35]: good_movies.sort(col("averageRating").desc(),col("numVotes").desc()).select("originalTitle", "startYear", "averageRati
         ng", "numVotes").show(5)

+--------------------+---------+-------------+--------+
|       originalTitle|startYear|averageRating|numVotes|
+--------------------+---------+-------------+--------+
|     The Dark Knight|     2008|          9.0| 1971593|
|The Lord of the R...|     2003|          8.9| 1425695|
|           Inception|     2010|          8.8| 1752024|
|The Lord of the R...|     2001|          8.8| 1442541|
|The Lord of the R...|     2002|          8.7| 1288857|
+--------------------+---------+-------------+--------+
only showing top 5 rows
```

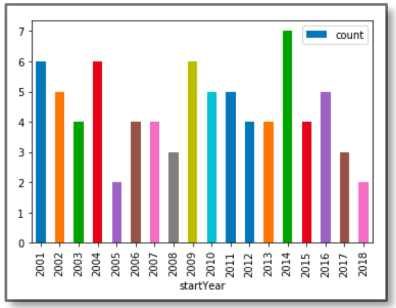
Solution

```
In [36]: good_movies.sort(col("averageRating").desc(),col("numVotes").desc()).select("originalTitle", "startYear", "averageRati
         ng", "numVotes").write.format("csv").save("hdfs:///user/hadoop/good_movies")
         # saved on HDFS as /user/hadoop/good_movies/part-00000-f01c02b6-516a-4a28-9b7e-d271bfc47536-c000.csv
         ...
```

d) How many movies are in list of c)?

```
In [37]: print good_movies.count()
         79
```

e) create following plot with result of c) (plot visualizes the amount of good movies per year since 2001)



Solution

```
In [38]: plot_dataframe = good_movies.groupBy('startYear').count().sort(col("startYear").asc())
```

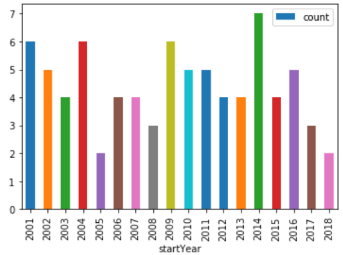
```
In [39]: plot_dataframe.show()
```

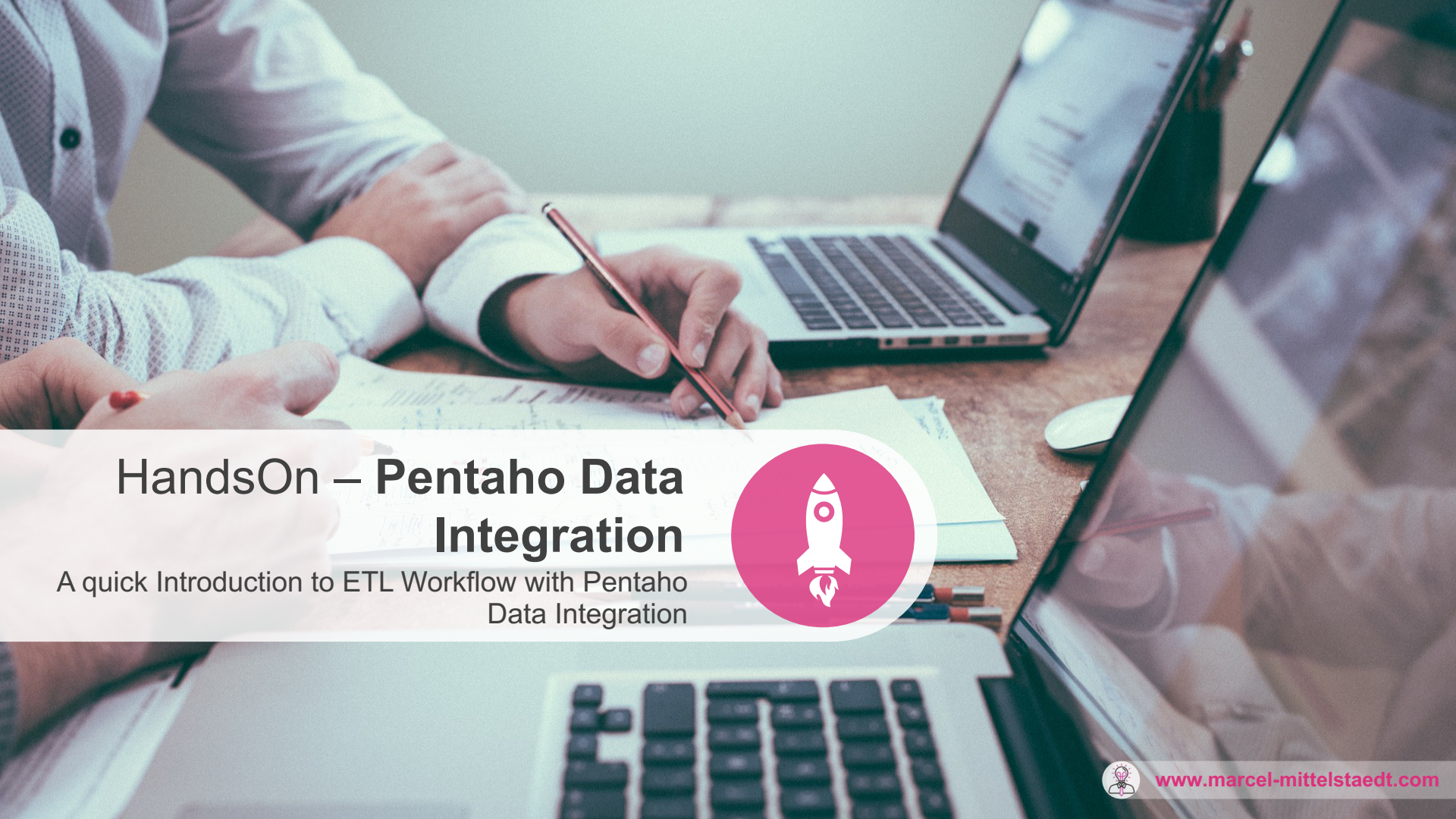
```
+---------+-----+
|startYear|count|
+---------+-----+
|     2001|    6|
|     2002|    5|
|     2003|    4|
|     2004|    6|
|     2005|    2|
|     2006|    4|
|     2007|    4|
|     2008|    3|
|     2009|    6|
|     2010|    5|
|     2011|    5|
|     2012|    4|
|     2013|    4|
|     2014|    7|
|     2015|    4|
|     2016|    5|
|     2017|    3|
|     2018|    2|
+---------+-----+
```

```
In [40]: import matplotlib.pyplot as plt
         import pandas
         pd_df=plot_dataframe.select("startYear", "count").toPandas()
```

```
In [41]: pd_df.plot.bar(x='startYear',y='count')
```

```
Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe72495ec50>
```

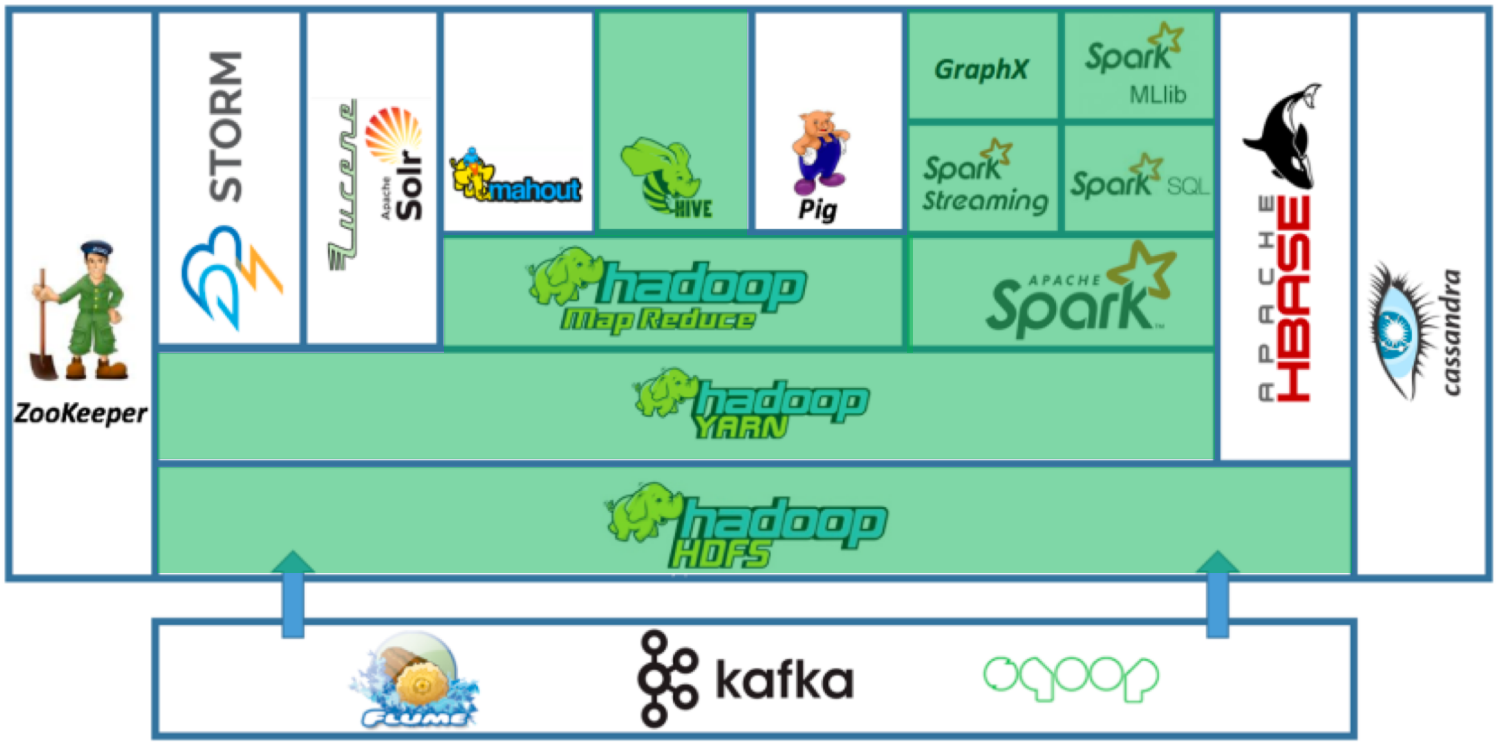




HandsOn – **Pentaho Data Integration**

A quick Introduction to ETL Workflow with Pentaho Data Integration

The Hadoop Ecosystem



Today's (exercise) focus

Exercises Preparation I

Install and Setup Pentaho Data Integration

Download And Install PDI

1. Download Pentaho Data Integration (8.1.0)

```
https://sourceforge.net/projects/pentaho/
```

2. Extract and Move:

```
unzip pdi-ce-8.1.0.0-365.zip
```

```
mv data-integration /opt/pdi
```


Start PDI (Spoon)

3. Start Pentaho Data Integration

UNIX:

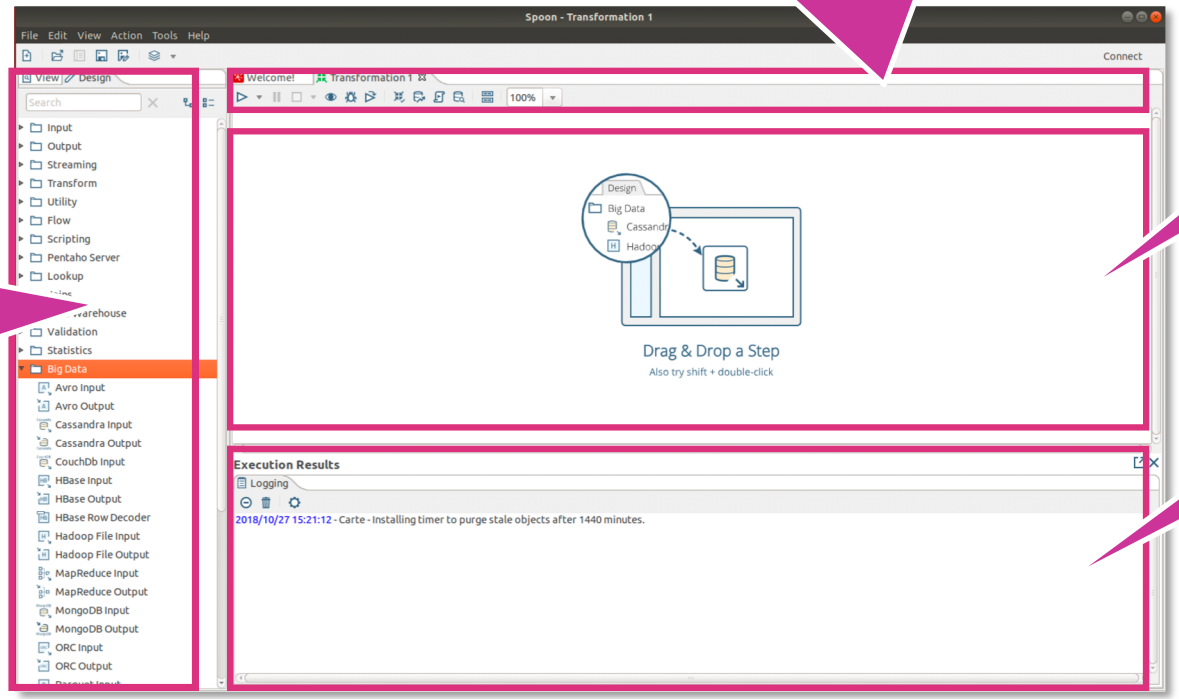
```
/opt/pdi/spoon.sh
```

Windows:

```
/some/directory/spoon.bat
```


Spoon Interface

3. Start Pentaho Data Integration



Transformation/Job Toolbar

Work/Design Area

Steps for Jobs and Transfor-mations

Execution Results/Logfile

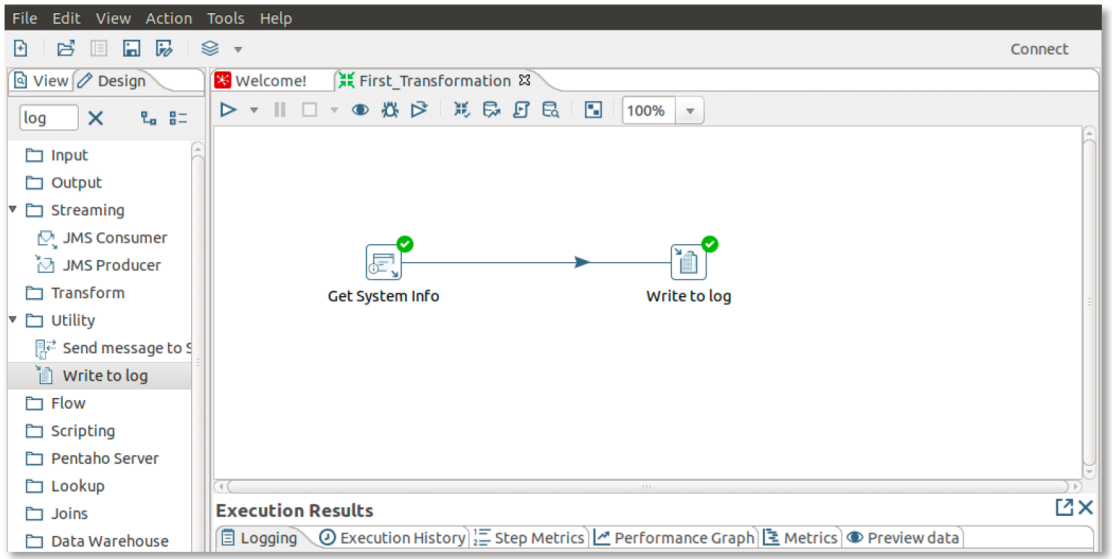
Exercises Preparation II

Some naive ETL examples of using Pentaho Data Integration

PDI Basics/Examples – First Transformation

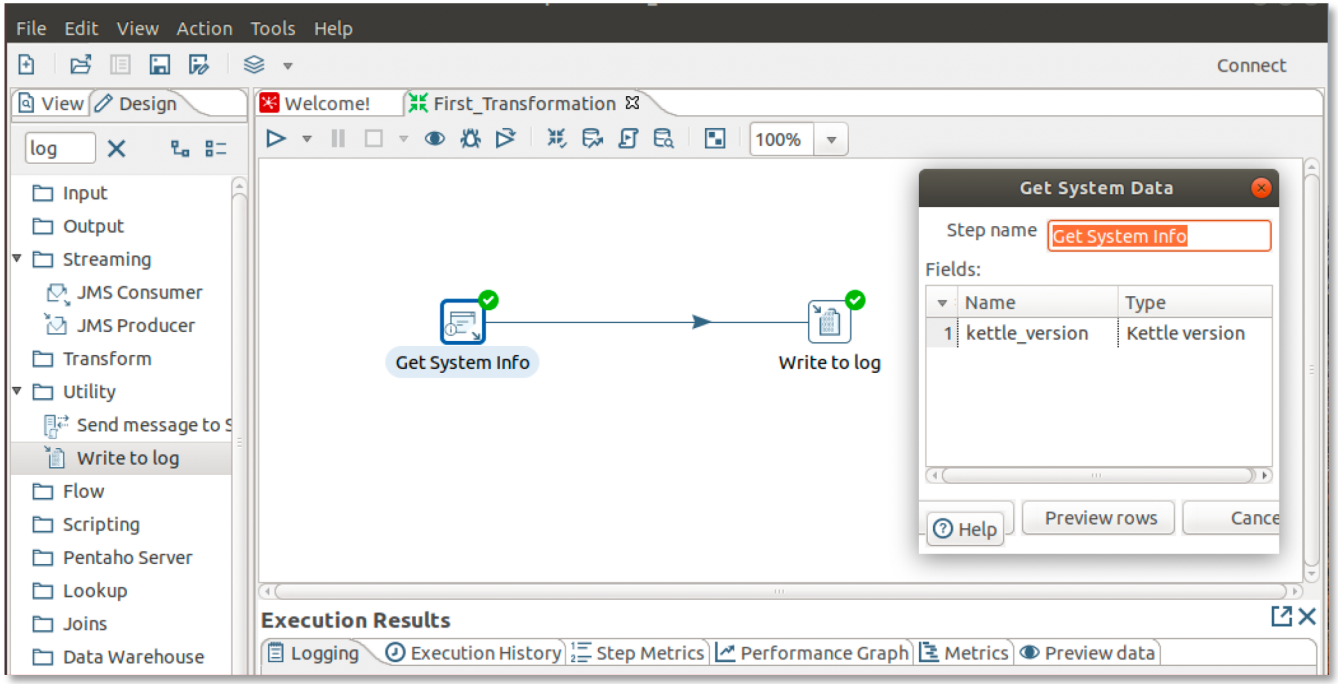
1. Create new **Transformation** (Click: *File → New → Transformation*)

2. Drag&Drop step „*Get System Info*“ and „*Write To Log*“ into Work/Design Area and connect both steps:



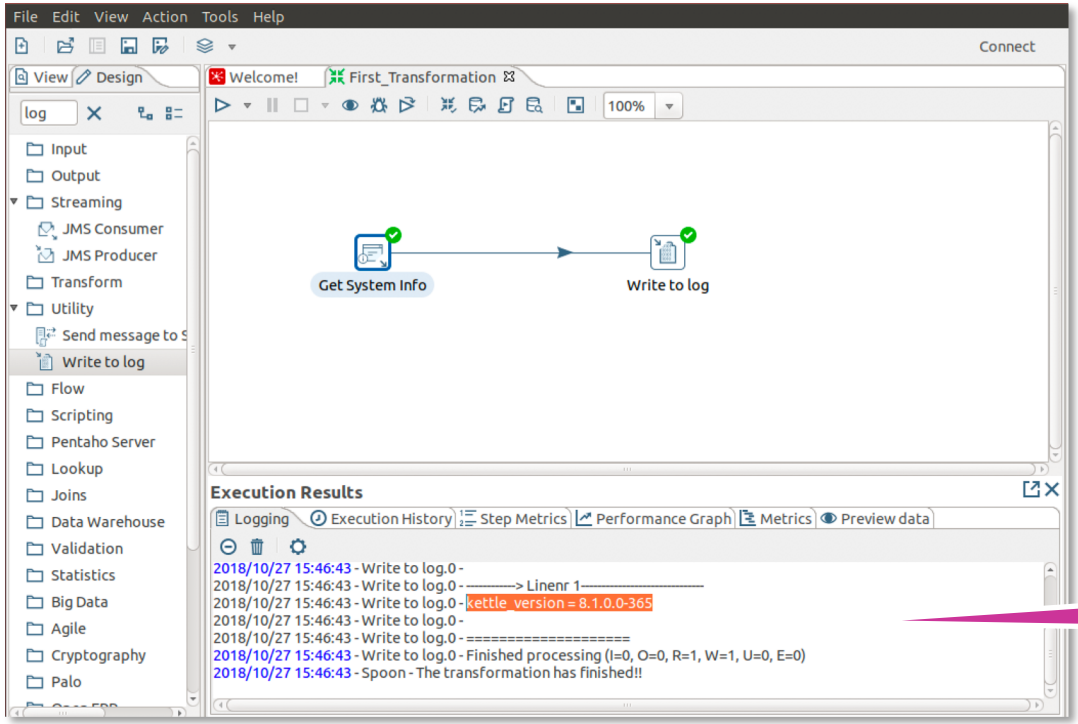
PDI Basics/Examples – First Transformation

3. Double Click „*Get System Info*“ step to configure step accordingly to kettle (PDI) version info:



PDI Basics/Examples – First Transformation

4. Save Transformation (*First_Transformation*). Press Run Button. See Results:



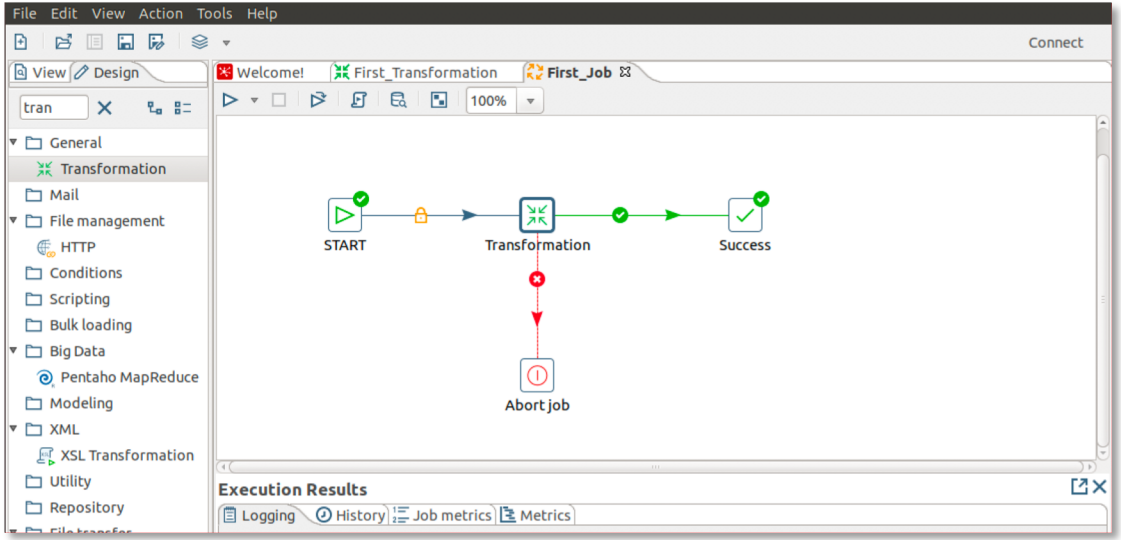
- results of „*Get System Info*“ step are redircected to „Write To Log“ step
- „Write to Log“ step will write results to execution log
- **transformations** are all about actual data transform ation and data flow

Results

PDI Basics/Examples – **First Job**

1. Create New **Job** (Click: *File → New → Job*)

2. Drag&Drop step „*START*“, „Transformation“, „Abort Job“ and „Success“ steps into Work/ Design Area and connect all steps accordingly:



- Start of each Job

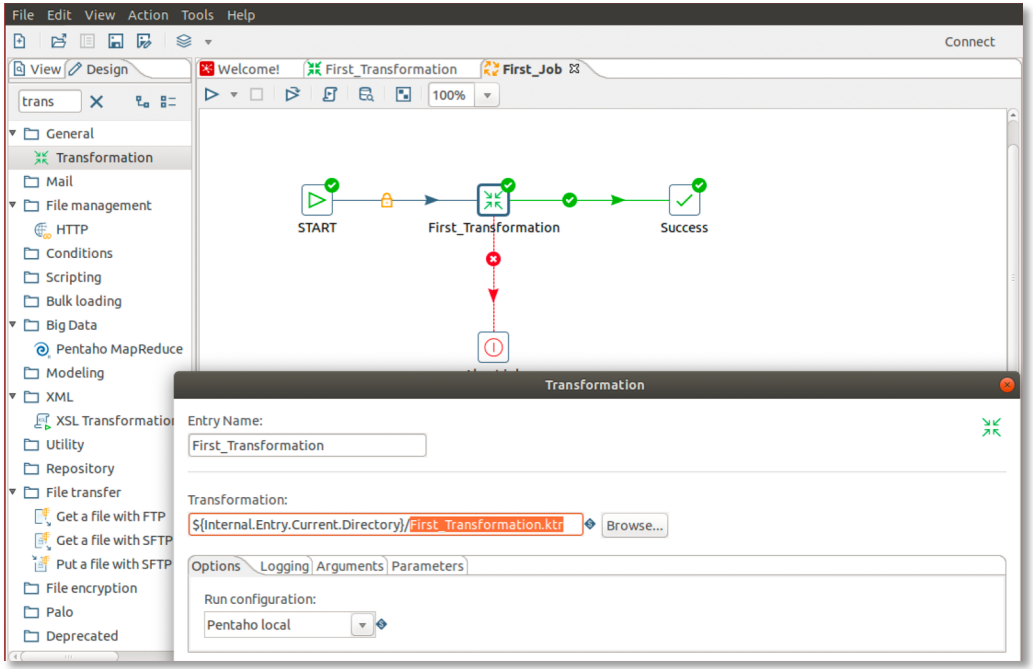
- End of a Job (if not successful)

- End of a Job (if successful)

- Includes a Transformation

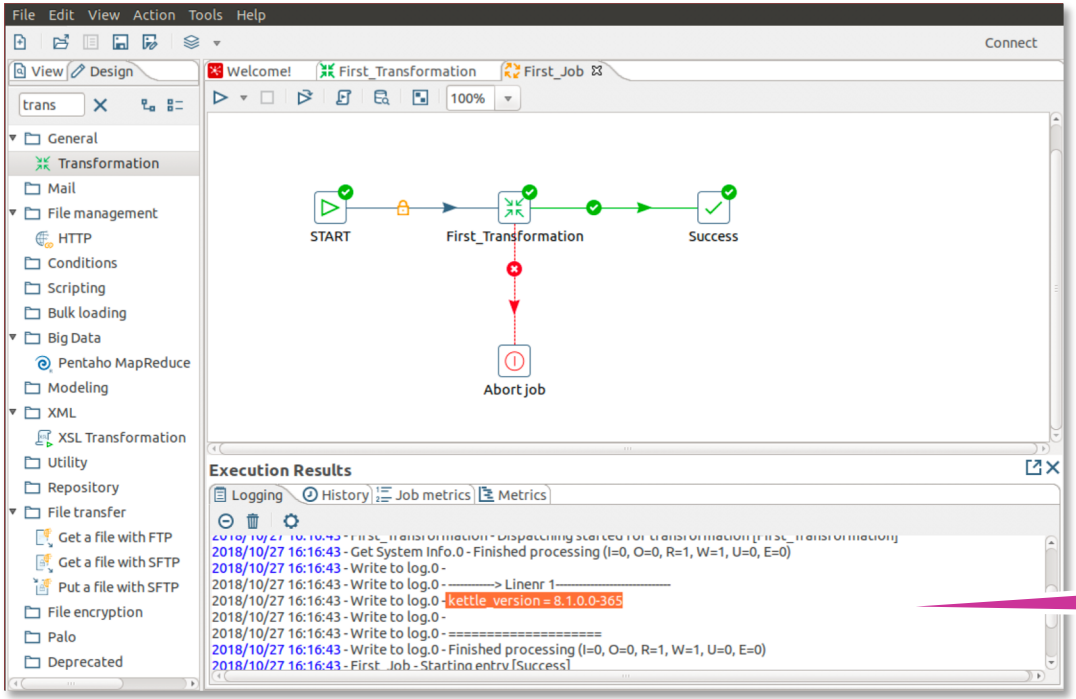
PDI Basics/Examples – **First Job**

3. Include previously created Transformation (First_Transformation) by double click on step „*Transformation*“ and including „*First_Transformation.ktr*“:



PDI Basics/Examples – **First Job**

4. Save Job (*First_Job*). Press Run Button. See Results:



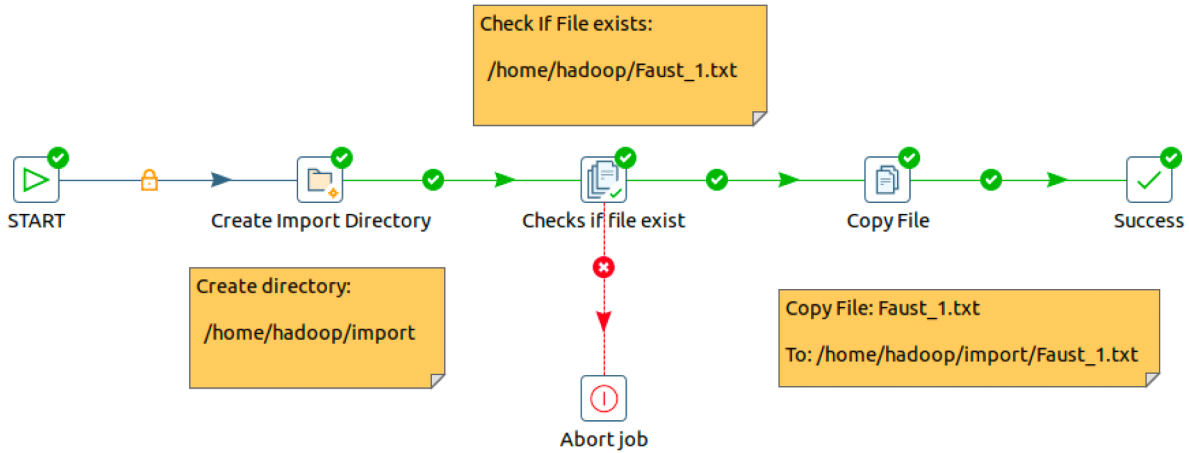
- Job will execute previously created transformation (*First_Transformation.ktr*)
- Output of transformation will be appended to execution log of Job
- **jobs** are all about workflows of multiple transformations and jobs

Results

PDI Basics/Examples – **Local Filesystem**

1. Create New **Job** *Local_FileSystem.kjb* using steps:
   - „*Create a Folder*“,
   - „*Check if files exist*“ and „*Copy File*“

…to check wether `/home/hadoop/Faust_1.txt` exists and either abort (if not) or copy it to import directory `/home/hadoop/import/Faust_1.txt`



PDI Basics/Examples – **HDFS**

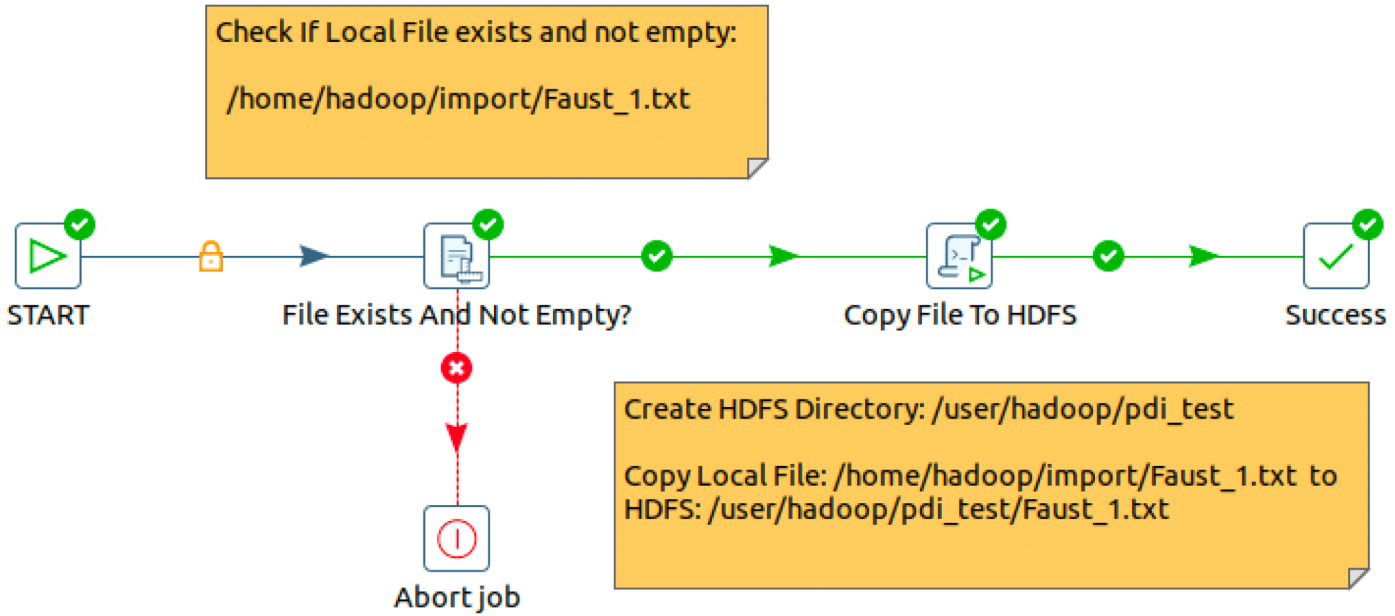
1. Start HDFS and Yarn:

```
start-dfs.sh
start-yarn.sh
```

2. Create New **Job** *HDFS_Filesystem.kjb* using steps:
   „*Evaluate file metrics*“ – to check wether file exists and is not empty
   „*Shell*“ – to execute bash commands (e.g. **`hadoop fs -put`** … in our case)

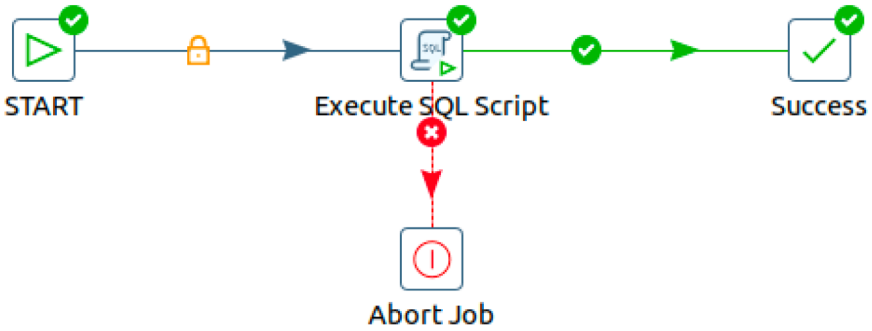
PDI Basics/Examples – **HDFS**

3. Save and run job:



PDI Basics/Examples – **Hive**

1. Create New **Job** *Hive_SQL.kjb* using step „*Execute SQL Script*“:

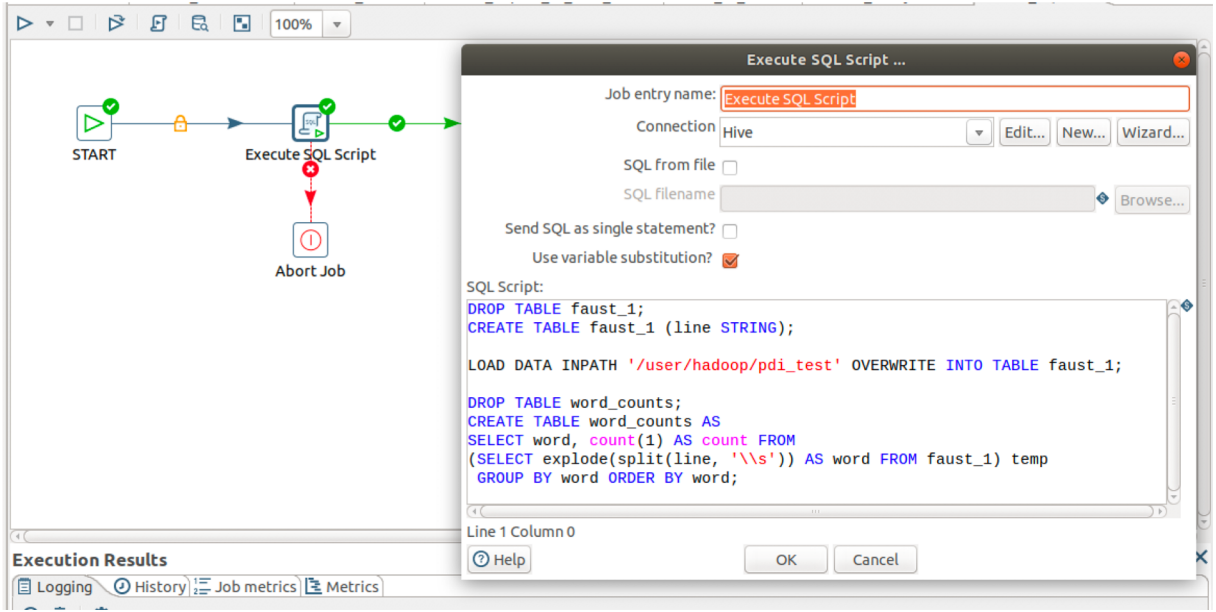


2. Start HiveServer2 (to enable job to connect to Hive):

```
/home/hadoop/hive/bin/hiveserver2
```

PDI Basics/Examples – **Hive**

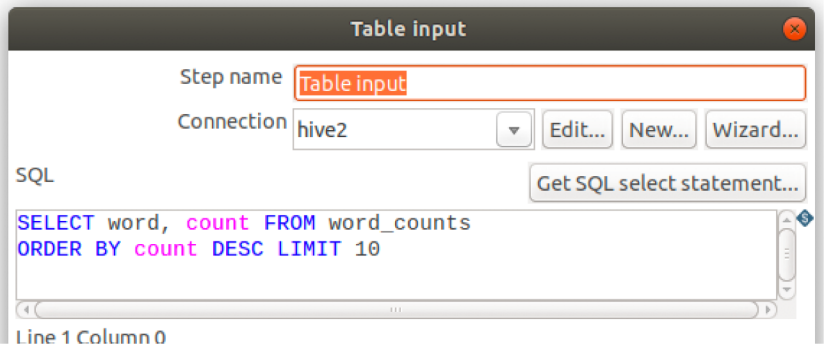
3. Add HiveQL statement to step „*Execute SQL Script*“ (word count example of previous le cture) and run job:



- Job will create table `faust_1`
- Load file *Faust_1.txt* from previous step into table `faust_1`
- Calculate word counts and save results to table `word_counts`

PDI Basics/Examples – **Parameters**

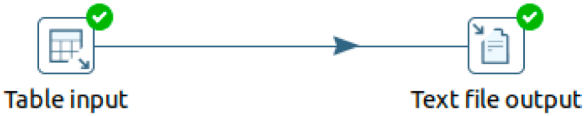
1. Create New **Transformation** *WordCount_Parameter.ktr* using steps *„Table Input“* and *„Text File output“*:

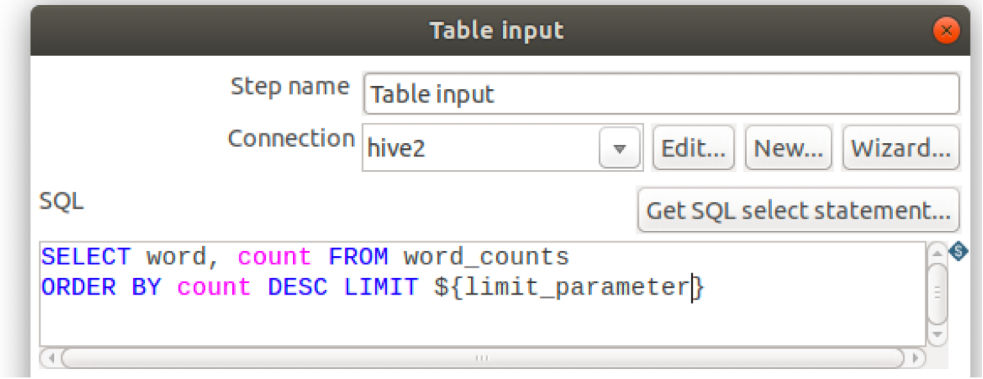


- *„Table Input“* reads word count results from table `word_counts` from previous step
- „*Text file output*“ saves result (Top 10 word counts) of „*Table input*“ query to csv file on local filesystem

PDI Basics/Examples – **Parameters**

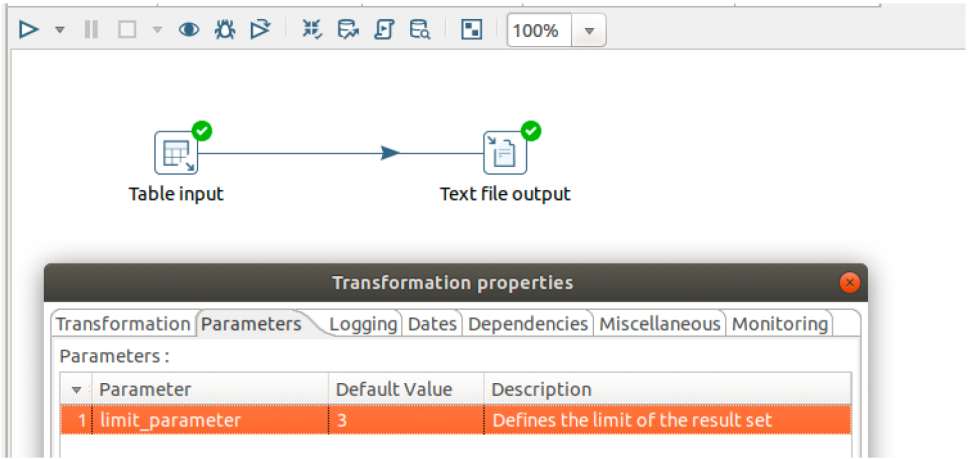
2. Stop! **Parameters?** We will make use of a parameter to decide how many results we want to have inside the result set (local CSV file):





PDI Basics/Examples – **Parameters**

3. Parameters can be set using Edit → Settings → Parameters



4. Run job. Output CSV will only have 3 entries now.

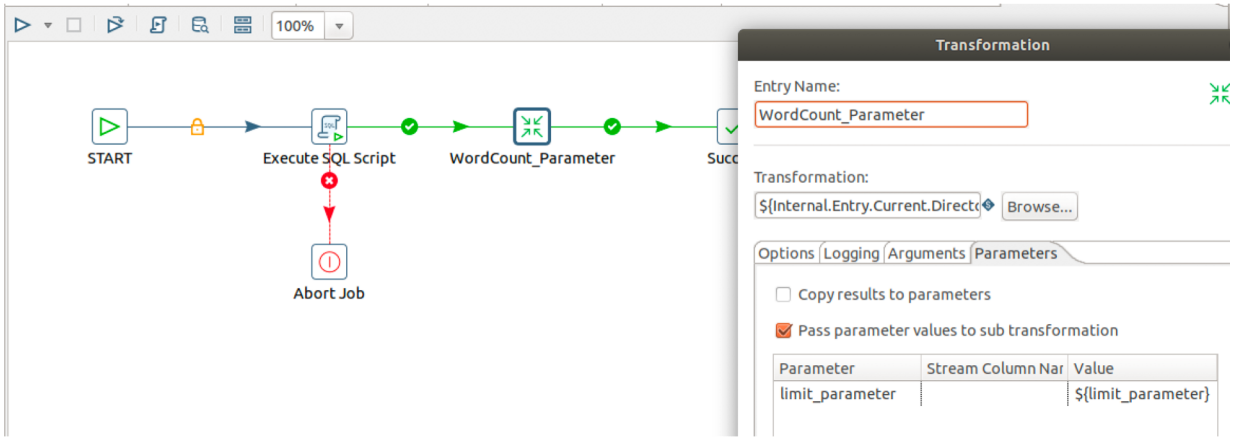
```
hadoop@marcel-VirtualBox:~$ cat export/word_count.csv
word,count
          ,1603
und       ,509
die       ,463
```

PDI Basics/Examples – **Parameters**

5. Parameters can also be set within kettle.properties file:

```
hadoop@marcel-VirtualBox:~$ vi .kettle/kettle.properties
limit_parameter=7
```

6. Parameters can also passed from Job To Job/Transformation, e.g. by extending previous example job „*Hive_SQL.kjb*“ by transformation „*WordCount Parameter.ktr*“ and passing kettle.properties parameter `limit_parameter` to it:



PDI Basics/Examples – **Execute Jobs Using Kitchen**

1. It‘s nice to run jobs locally during development, but how to run them on a remote server (e.g. in a productive manner)? This is easily achieved using kitchen.sh:

```
/opt/pdi/kitchen.sh -file=/home/marcel/HDFS_Filesystem.kjb

[...]
2018/10/27 20:41:05 - HDFS_Filesystem - Start of job execution
log4j:ERROR No output stream or file set for the appender named [pdi-execution-appender].
2018/10/27 20:41:05 - HDFS_Filesystem - Starting entry [File Exists And Not Empty?]
2018/10/27 20:41:05 - HDFS_Filesystem - Starting entry [Copy File To HDFS]
2018/10/27 20:41:05 - Copy File To HDFS - Running on platform : Linux
2018/10/27 20:41:09 - HDFS_Filesystem - Starting entry [Success]
2018/10/27 20:41:09 - Carte - Installing timer to purge stale objects after 1440 minutes.
2018/10/27 20:41:09 - HDFS_Filesystem - Finished job entry [Success] (result=[true])
2018/10/27 20:41:09 - HDFS_Filesystem - Finished job entry [Copy File To HDFS] (result=[true])
2018/10/27 20:41:09 - HDFS_Filesystem - Finished job entry [File Exists And Not Empty?] (result
=[true])
2018/10/27 20:41:09 - HDFS_Filesystem - Job execution finished
2018/10/27 20:41:09 - Kitchen - Finished!
2018/10/27 20:41:09 - Kitchen - Start=2018/10/27 20:41:00.243, Stop=2018/10/27 20:41:09.265
2018/10/27 20:41:09 - Kitchen - Processing ended after 9 seconds.
```

Exercises Preparation III

A simple ETL examples of how to use an ETL Workflow tool (PDI) to integrate IMDb data

PDI IMDb Import

Previous examples have been nice to get a basic understanding of how PDI works, but not in a productive manner, as:

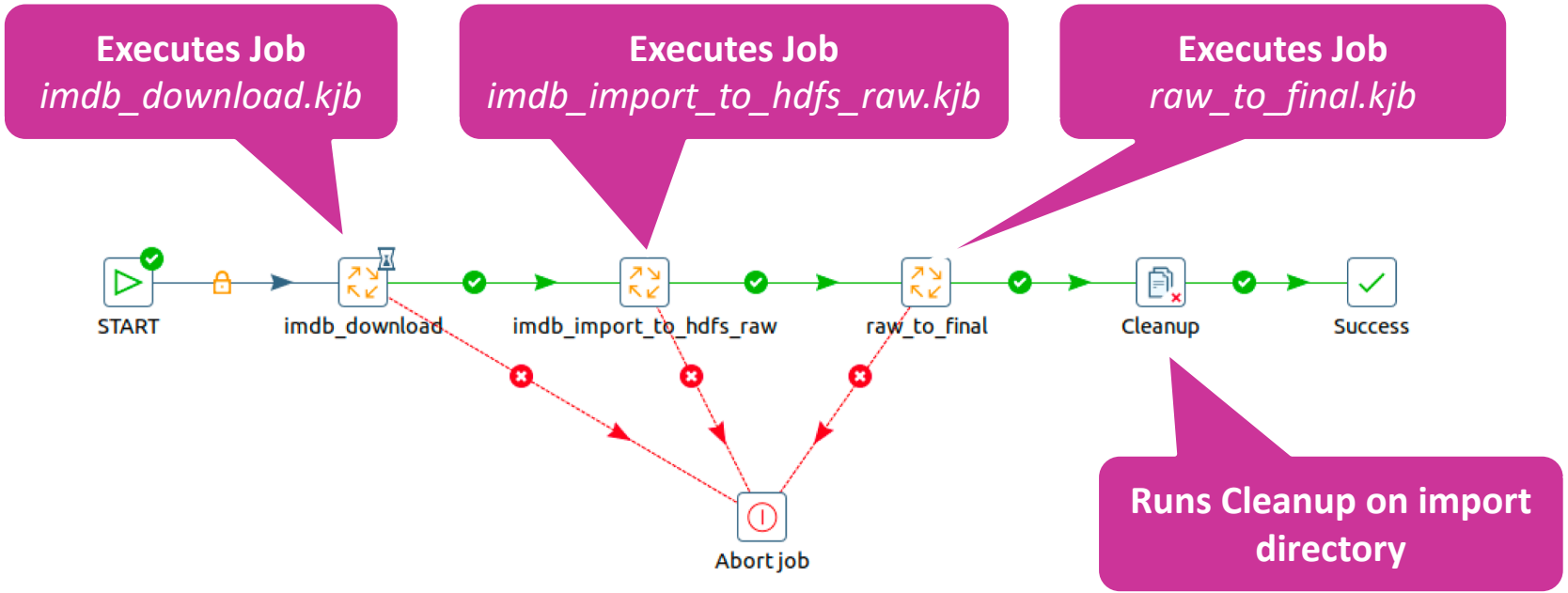
- There haven‘t been any complex workflows
- An appropriate usage of parameters
- Dependencies
- …

Thats why we take a look at a real world example: **IMDB data**

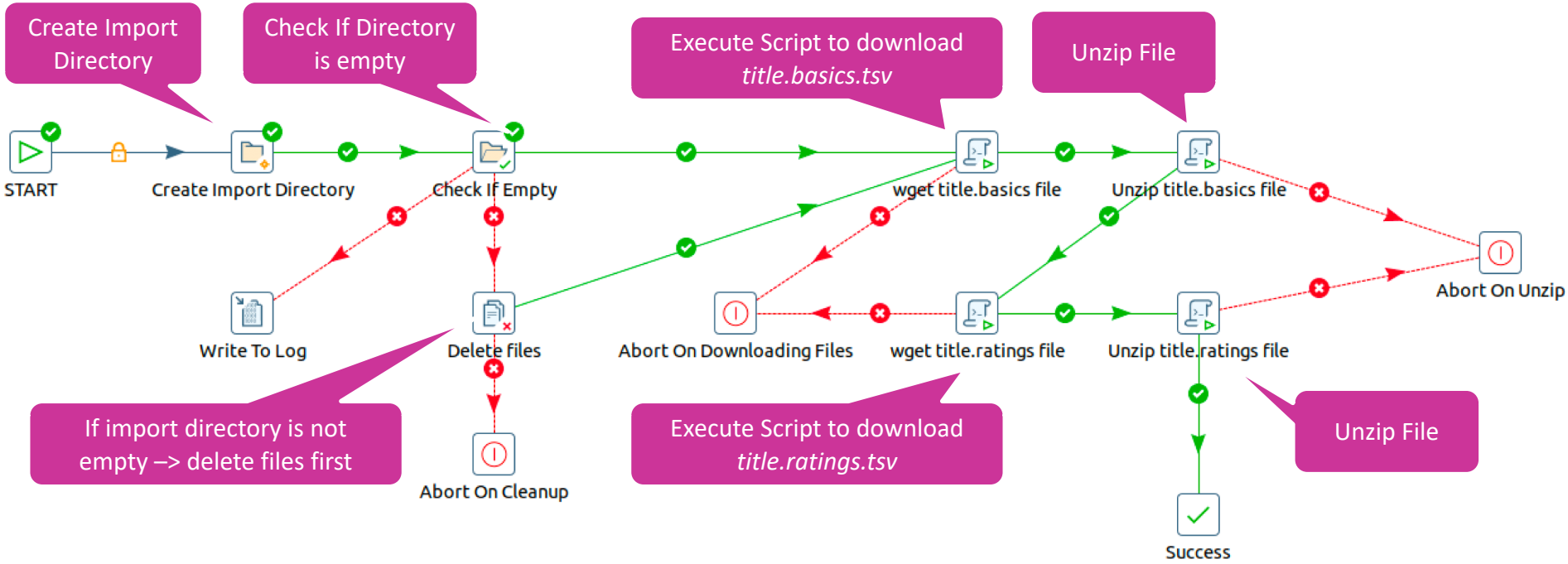
And Describe a complete workflow which is able to run each day:

- **Download of IMDb data** to local filesystem
- Move Data to HDFS (raw directory/layer)
- Create Hive tables for **Raw Layer**
- Create and fill **Final Layer** (Hive tables) by raw layer tables, applying business rules and using dynamic partitioning

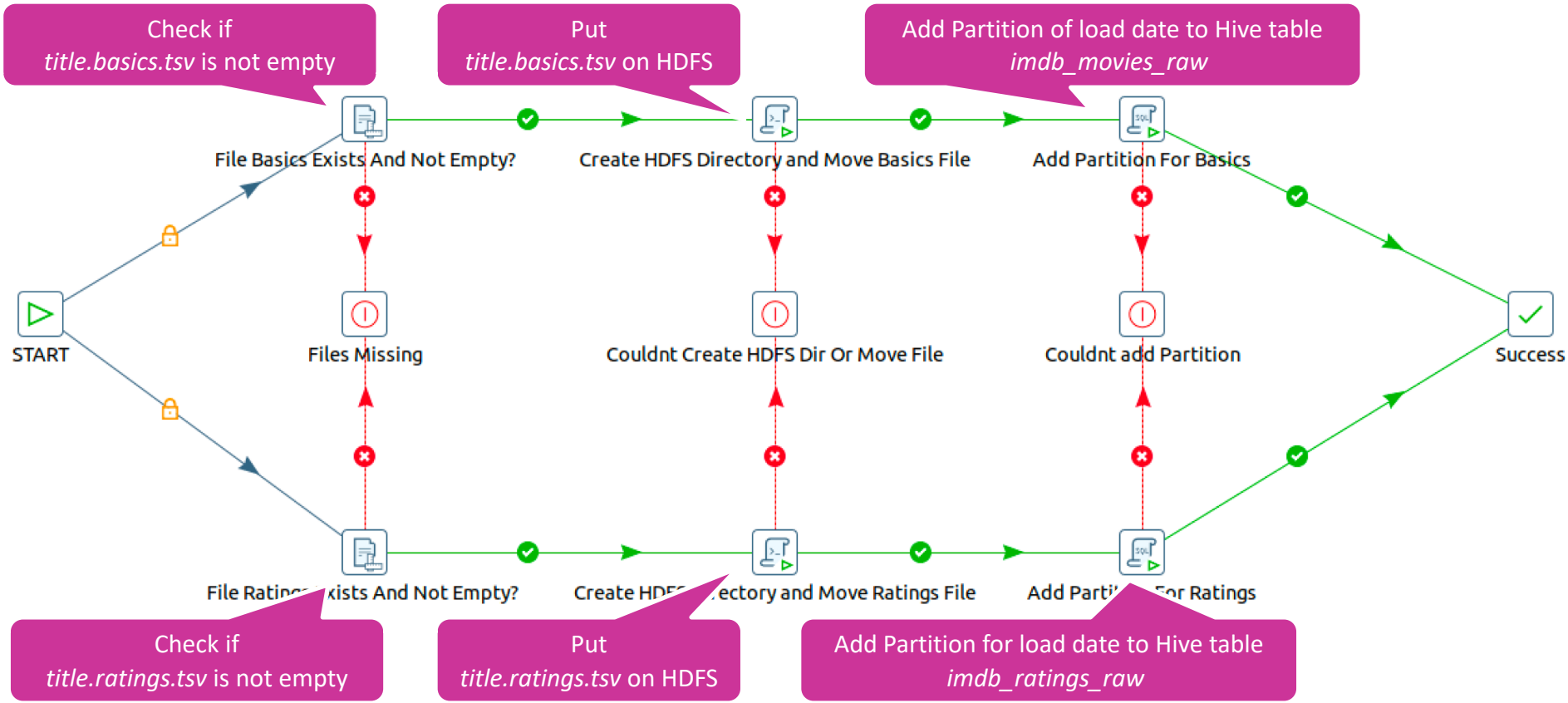
PDI IMDb Import – **Main Job**



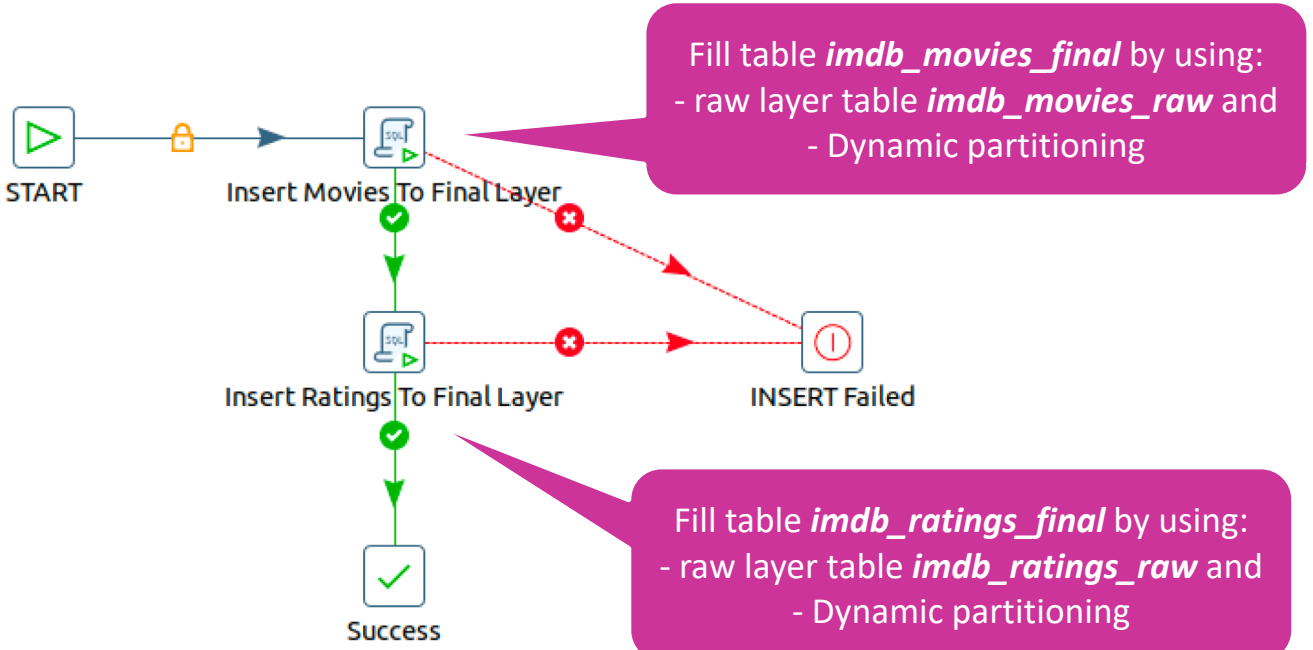
PDI IMDb Import – *imdb_download.kjb*



PDI IMDb Import – *imdb_import_to_hdfs_raw.kjb*



PDI IMDb Import – *raw_to_final.kjb*



PDI IMDb Import – **Execution**

1. Execute using Spoon or kitchen.sh:

```
/opt/pdi/kitchen.sh -file=/home/hadoop/imdb_main.kjb -param:load_day=29
-param:load_month=10 -param:load_year=2018
```

A Job like this could be scheduled by `cron` to run on a daily basis, receiving paramters:

- load_day
- load_month
- load_year

… from cron job

Exercises

Use Pentaho Data Integration to solve exercises based on IMDb data

Pentaho Data Integration Exercises – **IMDB**

1. Execute Tasks of previous HandsOn Slides

2. Use PDI and previous Jobs&Transformations to do following changes:

   a) **Extend job** *imdb_download.kjb* to also download ***name.basics.tsv.gz***

   b) **Extend job** *imdb_import_to_hdfs_raw.kjb* to also import ***name.basics.tsv*** to HDFS raw layer.

   c) **Create Hive table *imdb_actors_raw*** for ***name.basics.tsv*** in raw layer.
   Table should be partitioned by year, month and day of load date.

   d) **Create table *imdb_actors_final*** and **extend job *raw_to_final.kjb*** to also fill table *imdb_actors_final* using:
   - data of table *imdb_actors_raw* and
   - **partition table** by column *partition_is_alive* containing „alive“ or „dead“, wether the actor is alive or dead.

Pentaho Data Integration Exercises – **IMDB**

3. Run main workflow job **imdb_main.kjb** using:

```
/opt/pdi/kitchen.sh -file=/home/hadoop/imdb_main.kjb -param:load_day=29
-param:load_month=10 -param:load_year=2018
```