

Spark – Dataset

Apache Hive wykorzystywany jest bardzo często w roli hurtowni danych. Trwałość danych, kolumnowy format, zaawansowane mechanizmy analityczne, możliwość wykorzystania SQL bardzo w tym pomagają.

Nie ma jednak hurtowni danych bez procesów, które tę hurtownię będą zasilają. I tu pojawia się kolejne zastosowanie Sparka i Scali. Razem mogą stanowić bardzo udany tandem przy budowie procesów ETL.

Twoim zadaniem, tym razem mającym postać projektu, będzie:

1. Zaprojektowanie hurtowni danych w Apache Hive
2. Zaimplementowanie procesów ETL mających postać notatek Zeppelin wykorzystujących Scalę i Sparka (oczywiście w praktyce nasze procesy ETL powinny mieć charakter programów wykonywalnych *.jar rejestrowanych do powtarzalnego i bezobsługowego wykonywania)
3. Utworzenie notatek dla wybranych analiz

Opis projektu

W ramach projektu należy dokonać integracji danych źródłowych pochodzących z

- różnych systemów informatycznych (baz danych) i
- źródeł zewnętrznych (plików)

w celu umożliwienia przeprowadzenia szeregu analiz.

Aby tego dokonać należy wykonać trzy zadania:

1. Zaprojektować Hurtownię Danych w sposób umożliwiający dokonanie wymaganych analiz
2. Zaimplementować transformacje integrujące dane w Hurtowni Danych pochodzące z systemów informatycznych oraz danych zewnętrznych, a następnie dokonać integracji poprzez uruchomienie zaprojektowanych transformacji
3. Zaimplementować zapytania dokonujące wymaganych analiz na podstawie zawartości Hurtowni Danych

Wymagane analizy

1. Analiza/porównanie sprzedaży w poszczególnych regionach w poszczególnych latach dla wybranych kategorii produktów

Przykładowy ekran raportu:

Analiza sprzedaży, Kategoria produktu: Sci-Fi					
Lata		2013	2014	2015	2016
Regiony					
Central		5214,3	20140,6	25106,47	5301,46
East		11084,23	28945,81	32398,25	8399,22
West		5096,45	15369,04	20485,24	4538,05

- Analiza/porównanie zysku (sprzedaż minus koszt) ze sprzedaży poszczególnych kategorii w poszczególnych latach dla wybranych typów dni (wolnych/roboczych).

Przykładowy ekran raportu:

Analiza zysku, dni wolne: T				
Lata	2013	2014	2015	2016
Kategorie				
Akcja	3177,63	10667,14	14615,68	2766,43
Alkohol	18,44	54,10	68,72	4,88
Dla dzieci	2624,20	6793,33	10465,28	2928,12
Dramat	2917,88	8966,65	10192,07	2248,35
Gry Video	1323,33	3901,40	4415,55	1099,91
Horror	1958,88	6374,17	8391,80	1405,03
Komedia	2891,90	10618,64	14714,03	3153,53
Komedia romantyczna	3170,10	9563,63	15536,09	2737,39
Napoje bezalkoholowe	97,33	116,05	178,95	38,71
Przekąski	65,17	175,53	179,80	61,73
Sci-Fi	2365,52	6408,21	8684,64	1744,84
Slodycze	72,79	145,48	155,35	31,48
Thriller	2979,07	11780,66	18053,23	3432,43

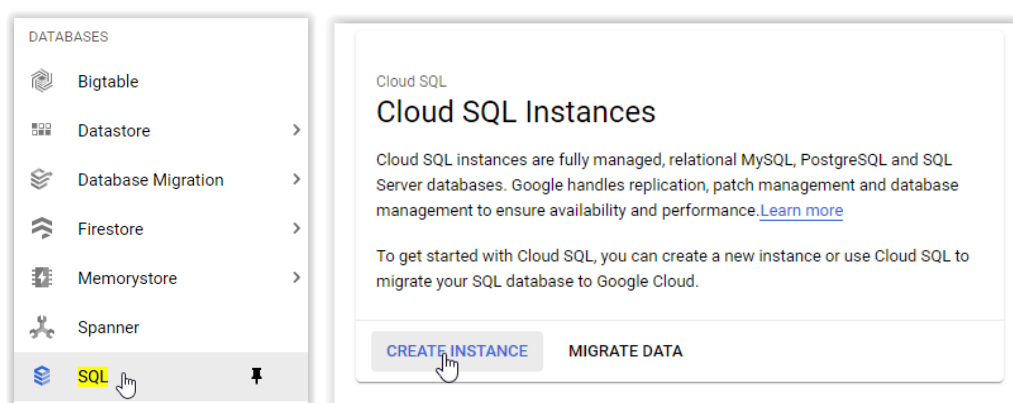
Przygotowanie środowiska

Instalacja baz danych East, West i Central

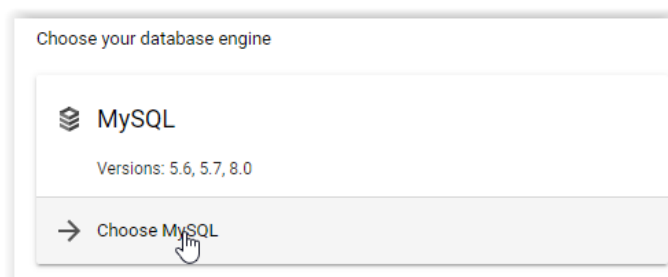
Naszymi źródłowymi systemami informatycznymi będą trzy bazy danych: **east**, **west** i **central** symulujące trzy rozproszone geograficznie bazy danych przedsiębiorstwa. W naszym przypadku będą to bazy danych mysql, ale dzięki temu, że zarówno Spark jak i notatniki Zeppelina pozwalają na podłączanie się do źródeł danych za pomocą JDBC, mogłyby to być praktycznie dowolne inne bazy danych.

Aby uprościć sobie zadanie utworzymy instancję mysql w środowisku Google Cloud Platform

- Z lewego menu konsoli Google Cloud Platform wybierz pozycję SQL z sekcji Databases, a następnie rozpocznij tworzenie nowej instancji



- Z dostępnych wersji wybierz MySQL



- Określ nazwę instancji, hasło np.: ewc, oraz region – ten sam, który wykorzystujesz podczas tworzenia klastra Dataproc. Wersję bazy danych ustaw na **5.7**. Zanotuj użyte hasło (PASS_MYSQL).

Instance info

Instance ID
Choice is permanent. Use lowercase letters, numbers and hyphens. Start with a letter.
east-west-central

Root password
Set a password for the root user. [Learn more](#)
... Generate

☐ No password

Location ?
For better performance, keep your data close to the services that need it.

Region
Choice is permanent
europe-west3 (Frankfurt)

Zone
Can be changed at any time
Any

- Rozwiń opcje konfiguracyjne tworzonej maszyny. Są one dość bogate, a my tego nie potrzebujemy. W sekcji *Machine type and storage*, skonfiguruj maszynę jak poniżej.

Machine type ?
For better performance, choose a machine type with enough memory to hold your largest table

Machine type	vCPUs	Memory
db-n1-standard-2	2	7.5 GB

[Change](#)

Storage type ?
Choice is permanent.

☐ SSD (Recommended)
Most popular choice. Lower latency than HDD with higher QPS and data throughput.

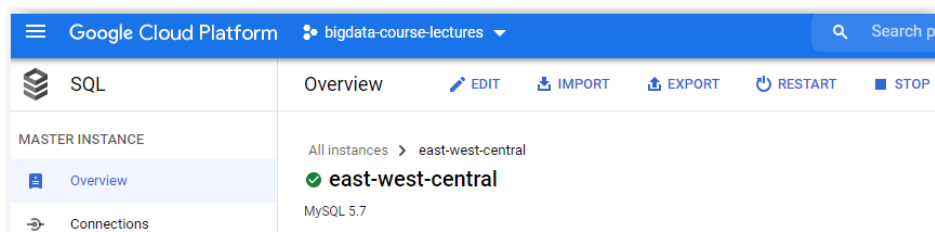
☒ HDD
Lower performance than SSD with lower storage rates.

Storage capacity ?
10 – 30720 GB. Higher capacity improves performance, up to the limits set by the machine type. Capacity can't be decreased later.

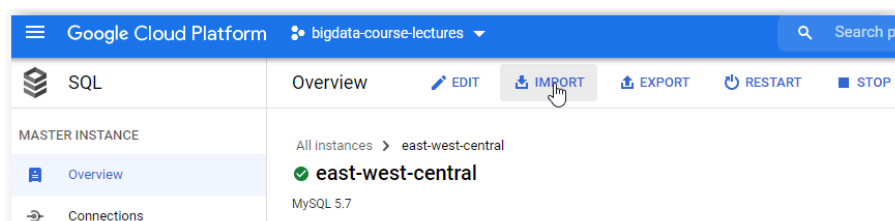
20 GB

☐ Enable automatic storage increases
If enabled, whenever you're nearing capacity, storage will be incrementally (and permanently) increased. [Learn more](#)

- Wybierz przycisk *Create* w celu utworzenia tak skonfigurowanej maszyny z instancją MySQL.



- Załaduj na swój zasobnik plik `mysql_east_west_central_dump.sql`
- Z konsoli maszyny MySQL wybierz przycisk *Import*



8. Wybierz z Twojego zasobnika dostępny tam plik Potwierdź swój wybór przyciskiem *Select*.



9. Wybierz przycisk *Import* aby rozpocząć import

← Import data from Cloud Storage

Source

Choose the file that you'd like to import data from
Browse for a file or enter the path for one (bucket/folder/file). Make sure you have read access first. [Learn more](#)

☒ big-data-course-bucket-2018-put/mysql_east_west_central_dump Browse

Indicate the format of the file that you're importing

☒ SQL
A plain text file with a sequence of SQL commands, like the output of mysqldump

☐ CSV
If your Cloud Storage file is a CSV file, select CSV. The CSV file should be a plain text file with one line per row and comma-separated fields.

Destination

Database ⓘ
Select a database only if your Cloud Storage import file does not specify any.

No database selected

Import

When you import, a Cloud SQL service account will be granted read access to your Cloud Storage file and the bucket that contains it. This will be reflected in your permissions.

10. Wybierz z lewego menu opcję *Databases*, aby potwierdzić powstanie trzech nowych baz danych

Name ↑	Collation	Character set	Type	
central	utf8_unicode_ci	utf8	User	⋮
east	utf8_unicode_ci	utf8	User	⋮
information_schema	utf8_general_ci	utf8	System	⋮
mysql	utf8_general_ci	utf8	System	⋮
performance_schema	utf8_general_ci	utf8	System	⋮
sys	utf8_general_ci	utf8	User	⋮
west	utf8_unicode_ci	utf8	User	⋮

11. Powróć na stronę główną tej konsoli i wybierz *Connect using Cloud Shell*. Przy okazji **zannotuj IP** naszej maszyny z MySQL (IP_MYSQL)

➡ Connect to this instance

Public IP address

35.246.141.10

Connection name

bigdata-course-lectures-187012:europa-west3:east-west-central

➡ Connect using Cloud Shell

Utworzenie klastra

1. Korzystając z poniższego polecenia i konsoli Cloud Shell utwórz klastę.

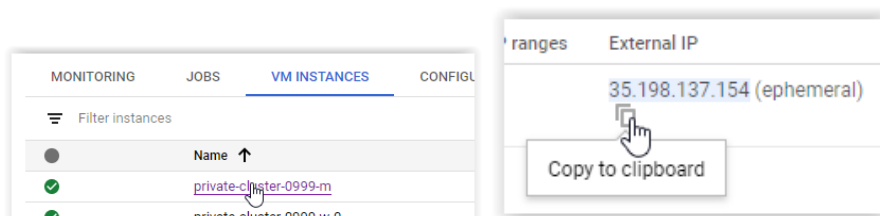
Podmień w poleceniu przed jego wykonaniem stosowne fragmenty.

UWAGA! Proszę zwrócić uwagę na obraz maszyny oraz komponenty dodatkowe.

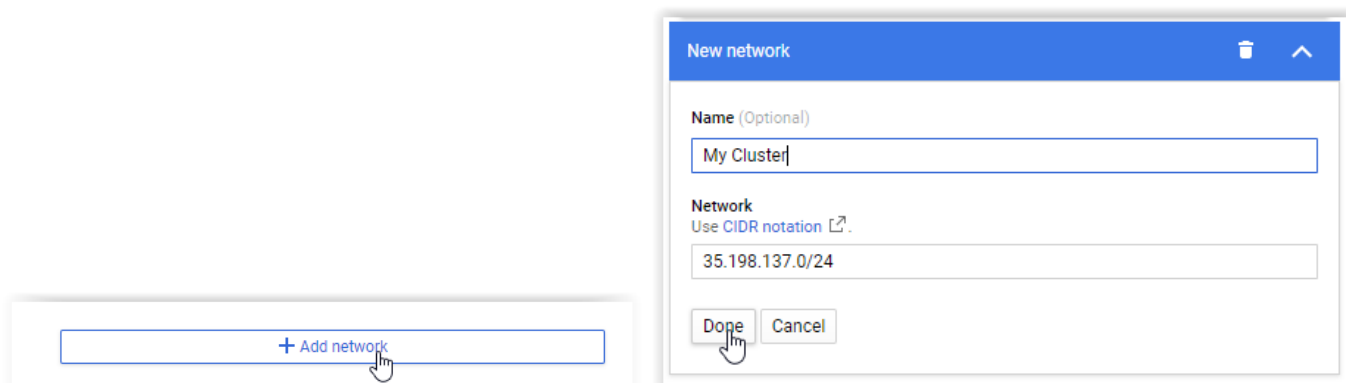
Czas uruchamiania maszyny ze względu na powyższe elementy definicji może być dłuższy.

```
CLUSTER_NAME={nazwa_klastra}
BUCKET_NAME={nazwa_uzywanego_zasobnika}
PROJECT_ID={id_projektu}
gcloud beta dataproc clusters create ${CLUSTER_NAME} \
  --enable-component-gateway --bucket ${BUCKET_NAME} \
  --region europe-west3 --zone europe-west3-b \
  --master-machine-type n1-standard-2 --master-boot-disk-size 50 \
  --num-workers 2 --worker-machine-type n1-standard-2 --worker-boot-disk-size 50 \
  --image-version preview-debian10 --optional-components ZEPPELIN \
  --project ${PROJECT_ID} --max-age=3h
```

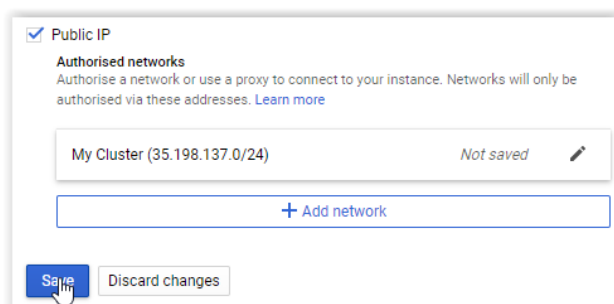
2. Otwórz terminal do serwera master utworzonego klastra.
3. Zanim opuścisz definicję klastra zaglądaj do definicji maszyny master. Skopiuj jej zewnętrzny adres IP



4. Powróć do definicji maszyny z MySQL, a następnie wybierz zakładkę *Connections*. Dodaj sieć obejmującą adres IP Twojego serwera master (i w domyśle pozostałych maszyn Twojego klastra).



5. Zapisz zmiany za pomocą przycisku *Save*



6. Będąc w terminalu maszyny master klastra sprawdź możliwość połączenia się z bazą danych MySQL.
Wykorzystaj do tego celu poniższe polecenie

```
mysql -h {IP_MYSQL} -u root -p
```

```
jankiewicz_krzysztof@private-cluster-0999b-m:~$ mysql -h 35.246.141.10 -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 717
Server version: 5.7.25-google-log (Google)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

7. Sprawdź ponownie jakie bazy danych już istnieją

```
show databases;
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| central |
| east |
| mysql |
| performance_schema |
| sys |
| west |
+-----+
7 rows in set (0.01 sec)
```

8. Sprawdź jakie tabele znajdują się w utworzonych w wyniku importu bazach danych.

```
select table_name, table_schema
from information_schema.tables
where table_schema in
('central','east','west');
```

```
mysql> select table_name, table_schema
-> from information_schema.tables
-> where table_schema in ('central','east','west');
+-----+-----+
| table_name | table_schema |
+-----+-----+
| etl_departamenty | central |
| etl_kategorie | central |
| etl_produkty | central |
| etl_sprzedaz | central |
| etl_departamenty | east |
| etl_kategorie | east |
| etl_produkty | east |
| etl_sprzedaz | east |
| etl_departamenty | west |
| etl_kategorie | west |
| etl_produkty | west |
| etl_sprzedaz | west |
+-----+-----+
12 rows in set (0.01 sec)
```

9. Możesz także sprawdzić strukturę i zawartość tabel. Wszystkie tabele o tych samych nazwach mają tą samą budowę. Zamknij połączenie z bazą danych MySQL za pomocą polecenia \q.

Łaďadowanie źródeł zewnętrznych

Naszymi danymi zewnętrznymi będą dwa pliki: `dni_wolne.xml` oraz `sklepy.txt`. Aby ułatwić sobie z nich korzystanie pobierz je (plik `ExternalData.zip`), a następnie załaduj je do systemu plików HDFS do katalogu `labs/spark/externaldata`

```
hadoop fs -copyToLocal gs://{nazwa_uzywanego_zasobnika}/ExternalData.zip
unzip ExternalData.zip
hadoop fs -mkdir -p labs/spark/externaldata
hadoop fs -copyFromLocal ExternalData/*.* labs/spark/externaldata
hadoop fs -ls labs/spark/externaldata
```

```
jankiewicz_krzysztof@private-cluster-0999-m:~$ hadoop fs -ls labs/spark/externaldata
Found 2 items
-rw-r--r--  2 jankiewicz_krzysztof hadoop      11205 2020-11-29 11:54 labs/spark/externaldata/dni_wolne.xml
-rw-r--r--  2 jankiewicz_krzysztof hadoop        752 2020-11-29 11:54 labs/spark/externaldata/sklepy.txt
```

Zestawy zadań projektowych

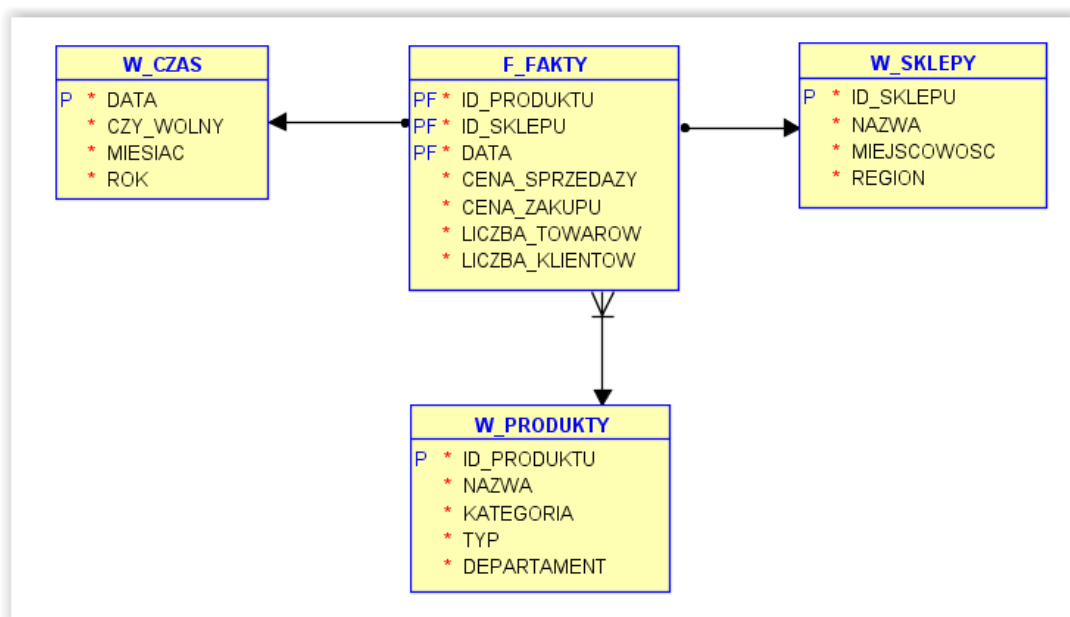
Zadanie 1 – zaprojektowanie Hurtowni Danych

1. Przeanalizuj
 - schematy źródłowych baz danych (east, west, central) oraz strukturę zewnętrznych danych źródłowych
 - analizy jakie mają być wynikiem końcowym projektu.
2. Na podstawie powyższych działań opracuj logiczny schemat Hurtowni Danych.
3. Jeśli masz jakiegokolwiek wątpliwości, prześlij schemat do weryfikacji przez prowadzącego zajęcia – popraw ewentualne błędy, uwzględnij przekazane uwagi.
4. Utwórz struktury danych w bazie danych Hive (dzięki funkcjonalności dostarczanej przez Apache Spark) odpowiadające zaprojektowanemu schematowi Hurtowni Danych.

Sugestie i odpowiedzi

- W efekcie powyższego zadania powinna zostać utworzona Hurtownia Danych w architekturze gwiazdy (choć nie jest to obligatoryjne) składająca się z tabeli faktów oraz trzech tabel wymiarów.

Przykładowy schemat mógłby wyglądać następująco:

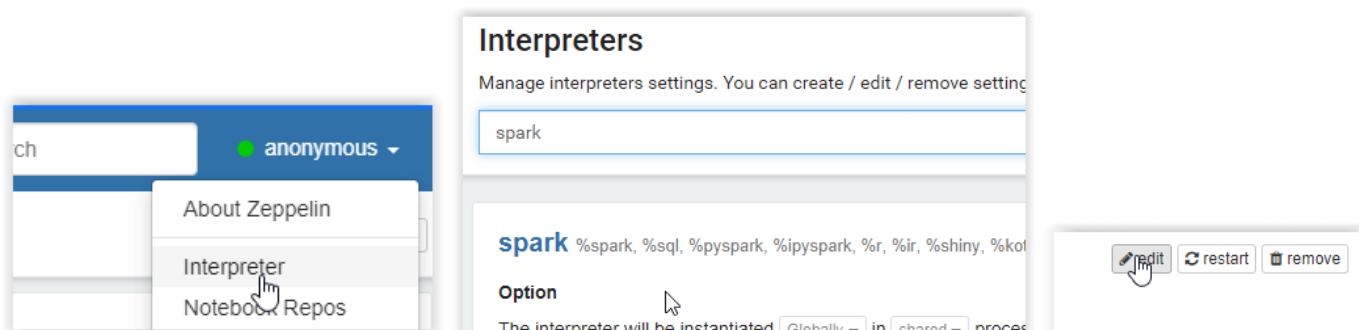


- W wielu konfiguracjach katalogi (dla repozytoriów danych) dla Apache Hive oraz Apache Spark są rozdzielone. Wynika to z faktu iż Apache Spark SQL wspiera tylko podstawowe funkcjonalności Apache Hive. Przykładowo brak wsparcia dla tabel typu *ACID*, Integracji z mechanizmem uwierzytelniania *Ranger*, *LLAP*. Taka sytuacja ma miejsce w wielu konfiguracjach (np. w maszynie HDP 3.0). To oczywiście nie oznacza, że te dwa produkty nie są zintegrowane. Integracja jest możliwa np. dzięki *Apache Spark - Hive Warehouse Connector* (ale to opowieść na inny czas). Powyższe oznacza, że aby uprościć implementację utwórz swoją hurtownię danych w katalogu Sparka – korzystając z mechanizmów Spark SQL.

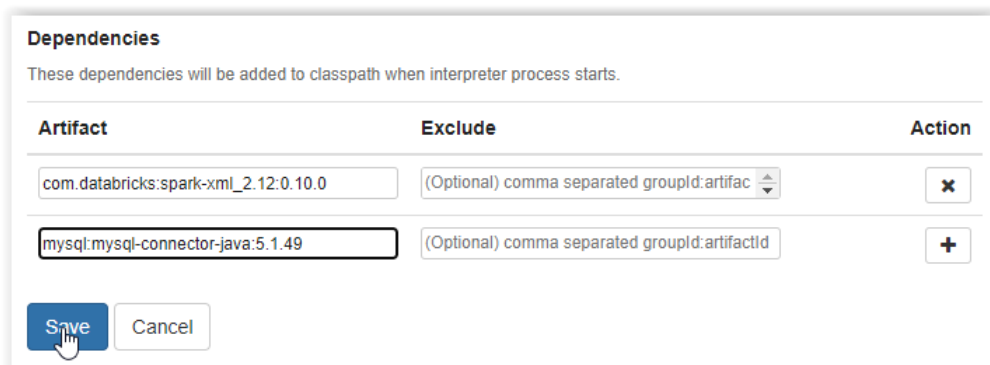
Rozwiązanie

W związku z tym, że notatki Zeppelina w środowisku Dataproc wykonywane są w ramach użytkownika zeppelin (ze względu na brak skonfigurowanego mechanizmu uwierzytelniania), cały nasz projekt, włącznie z etapem tworzenia struktur hurtowni danych, zostanie wykonany za ich pomocą.

1. Uruchom środowisko notatnika Zeppelina. Utwórz pierwszą z notatek o nazwie Create HD z domyślnym interpreterem spark.
2. Ze względu na funkcjonalności, które będziemy wykorzystywali w późniejszych etapach naszego projektu, musimy dodać dodatkowe zależności do naszego interpretera Spark. W tym celu wybierz z prawego menu opcję *Interpreter*, a następnie wyszukaj ustawienia interpretera Spark i włącz ich edycję



3. Na dole znajdziesz możliwość dodawania zależności. Pierwsza z nich (`com.databricks:spark-xml_2.12:0.10.0`) pozwoli w sposób cywilizowany obsłużyć dokumenty XML, druga (`mysql:mysql-connector-java:5.1.49`) umożliwi połączenie się z bazą danych MySQL. Wybór tych bibliotek wynika z wersji Scali, z której korzysta nasz Spark, oraz wersji MySQL. Wprowadź te zależności i zatwierdź zmiany przyciskiem *Save*.



Pobranie bibliotek z repozytorium Mavena może potrwać chwilę, zostaną one pobrane i wykorzystane podczas uruchamiania domyślnego interpretera czyli Sparka. Kolejne notatki, które będziemy tworzyli (o ile będziemy to robili w ramach tej samej sesji) będą korzystały z tego samego interpretera (tak jest skonfigurowany obecnie Zeppelin). Nie będzie więc potrzeby ponownego ładowania tych bibliotek.

4. Powrót do naszej notatki Create HD. W pierwszy paragrafie dowiedz się jakie masz dostępne bazy danych.

```
spark.sql("show databases").show()
```

```
spark.sql("show databases").show()
```

```
+-----+
|namespace|
+-----+
|  default|
+-----+
```

Took 29 sec. Last updated by anonymous at November 29 2020, 1:30:36 PM.

5. Dla uproszczenia domyślna baza danych będzie naszą bazą danych hd. Sprawdź jakie są tam dostępne tabele, a następnie wykonaj polecenia tworzące sugerowany zestaw tabel dla naszej hurtowni danych. Dogłębna analiza wymagań dotyczących oczekiwanych analiz, a także danych źródłowych, powinna prowadzić do wniosku, że zaproponowany poniżej schemat jest dobrym rozwiązaniem. Każde z poniższych poleceń możesz wykonać w oddzielnym paragrafie.

```
spark.sql("show tables").show()
```

```
spark.sql("""CREATE TABLE `w_czas` (
  `rok` int,
  `miesiac` int,
  `data` date,
  `czy_wolny` boolean)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'""")
```

```
spark.sql("""CREATE TABLE `w_produkty` (
  `nazwa_produkту` string,
  `nazwa_kategorii` string,
  `nazwa_depertamentu` string,
  `typ` string,
  `id_produkту` int)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'""")
```

```
spark.sql("""CREATE TABLE `w_sklepy` (
  `region` string,
  `miasto` string,
  `sklep` string,
  `id_sklepu` int)
ROW FORMAT SERDE
  'org.apache.hadoop.hive ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive ql.io.orc.OrcOutputFormat'""")
```

```
spark.sql("""CREATE TABLE `f_fakty` (
  `cena_sprzedazy` float,
  `cena_zakupu` float,
  `liczba_towarow` int,
  `liczba_klientow` int,
  `id_sklepu` int,
  `data` date,
  `id_produktu` int )
ROW FORMAT SERDE
  'org.apache.hadoop.hive ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive ql.io.orc.OrcOutputFormat'""")
```

6. Sprawdź czy wszystkie tabele zostały utworzone.

```
spark.sql("show tables").show()
```

```
spark.sql("show tables").show()
```

database	tableName	isTemporary
default	f_fakty	false
default	w_czas	false
default	w_produkty	false
default	w_sklepy	false

Took 1 sec. Last updated by anonymous at November 29 2020, 1:38:41 PM.

Zadanie 2 – transformacje zasilające zawartość Hurtowni Danych

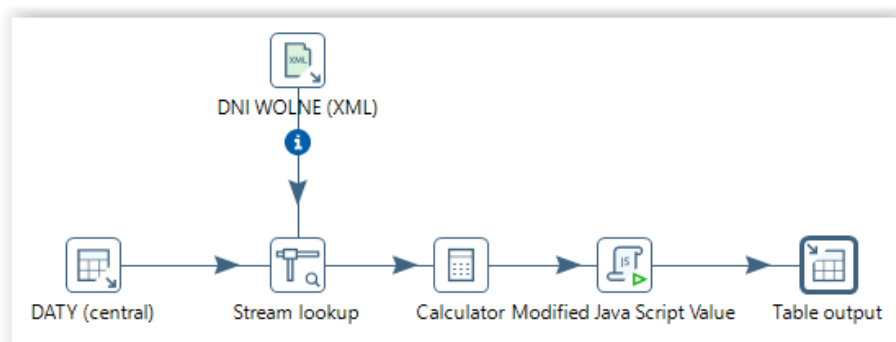
- Korzystając notatników Zeppelina utwórz transformacje
 - Zasilające tabele wymiarów Hurtowni Danych na podstawie zawartości danych źródłowych (3 transformacje – 3 notatki)
 - Zasilające tabele faktów Hurtowni Danych na podstawie zawartości (1 transformacja – 1 notatka)
- Wykorzystaj utworzone transformacje do zasilenia Hurtowni Danych na podstawie danych źródłowych

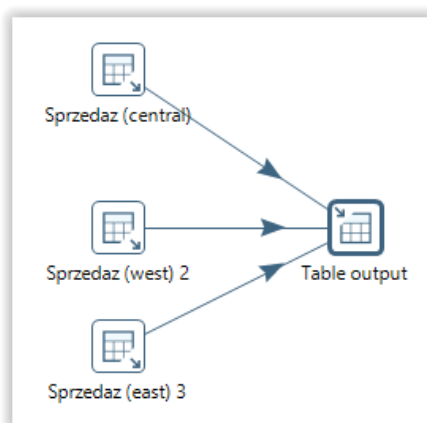
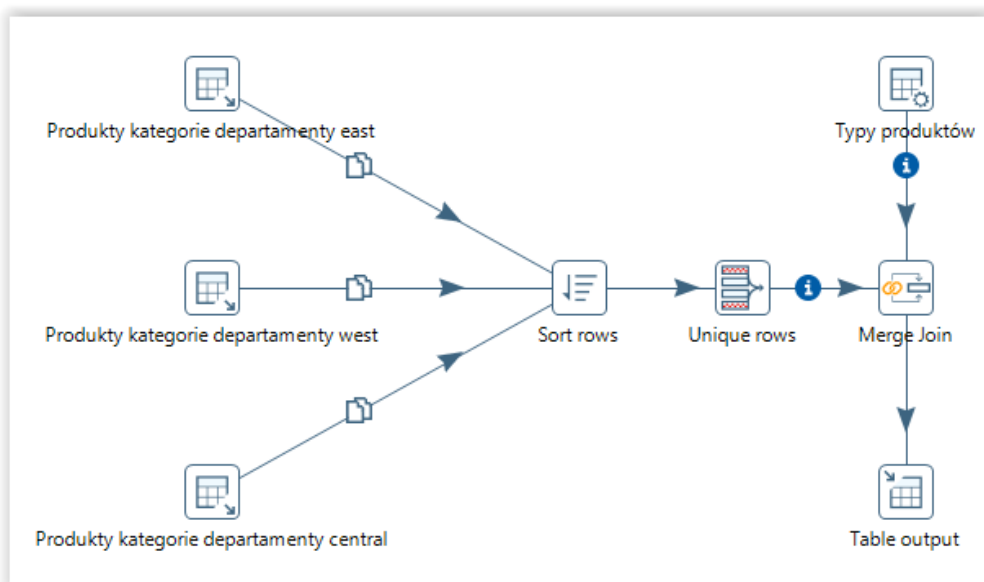
Wymagania, sugestie i podpowiedzi

- W sposób maksymalny z możliwych staraj się, na każdym etapie swojego przetwarzania, korzystać z typu Dataset i dostarczanych przez ten typ transformacji/akcji. Jest to obligatoryjny warunek pełnej akceptacji utworzonych transformacji.
- Niech każdy z paragrafów w swoich transformacjach będzie opatrzony tytułem, ewentualnie poprzedź każdy paragraf transformacji paragrafem %md, tak aby każdy etap przetwarzania był opatrzony odpowiednim komentarzem.
- Zasilamy oczywiście tabele w bazie danych Hive (baza danych hd).
- Obsługa plików źródłowych jest stosunkowo prosta. Pliki CSV przetwarzaliśmy wielokrotnie, z plikiem XML też sobie poradzisz. Poniżej znajdziesz przykładowy fragment kodu pozwalający na dostęp z poziomu notatnika Zeppelina do bazy danych mysql.

```
val kategorieDF = spark.read.format("jdbc").options(
  Map("url" -> "jdbc:mysql://{IP_MYSQL}:3306/east?autoReconnect=true&useSSL=false",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> "{PASS_MYSQL}",
    "dbtable" -> "etl_kategorie")).load()
```

- Jeśli chodzi o pliki XML, to postaraj się obsłużyć je profesjonalnie (czyli bez używania funkcji traktujących jego zawartość jak ciąg znaków np.: `substr` i `instr`). W celu uzyskania większej ilości informacji zaglądaj na: <https://github.com/databricks/spark-xml>
- Procesy ETL często realizowane przy pomocy specjalizowanych narzędzi np. Penatho DI. Poniżej znajdziesz zrzuty ekranów transformacji utworzonych właśnie w tym narzędziu. Mogą one być dla Ciebie sugestią dotyczącą tego jakie działania powinny być zawarte w Twoich transformacjach implementowanych przy wykorzystaniu Sparka i Scali





Jak widać nasze transformacje są naprawdę bardzo proste.

Rozwiązanie

Każda transformacja będzie miała postać oddzielnego notatnika. Nazywaj odpowiednio od nazw tabel, których dane są przez nie ładowane: w_sklepy, w_produkty, w_czas, f_fakty.

1. w_sklepy: Zaczniemy od transformacji dotyczącej **sklepów**. Jest zdecydowanie najprostsza. Wykorzystaj do jej zbudowania poniższe fragmenty kodu. Zwróć uwagę na prostotę zapisu danych do bazy danych Hive.

```
import org.apache.spark.sql._
val ip_mysql = "{IP_MYSQL}"
val pass_mysql = "{PASS_MYSQL}"
val username = "{twoj_username}"

val sklepy_DS = spark.read.format("org.apache.spark.csv").
  option("header", true).option("inferSchema", true).
  csv(s"/user/$username/labs/spark/externaldata/sklepy.txt").cache();
```

```
sklepy_DS.withColumnRenamed("SK_REGION", "region").
  withColumnRenamed("SK_MIEJSCOWOSC", "miasto").
  withColumnRenamed("SK_NAZWA", "sklep").
  withColumnRenamed("SK_ID", "id_sklepu").
  select("region", "miasto", "sklep", "id_sklepu").
  write.insertInto("w_sklepy");
```

2. w_produkty: Kolejnym wymiarem dla którego napiszemy transformację ETL będzie wymiar **produktów**. Zwróć uwagę na wykorzystanie trzech źródeł danych

```
import org.apache.spark.sql._

val eastProduktyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/east?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_produkty")).load()

val westProduktyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/west?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_produkty")).load()
```

```
val centralProduktyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/central?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_produkty")).load()
```

```
val allProduktyDF = centralProduktyDF.
  union(eastProduktyDF).
  union(westProduktyDF).
  dropDuplicates(Array("p_id"));
```

```
val eastKategorieDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/east?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_kategorie")).load()
```

```
val westKategorieDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/west?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_kategorie")).load()
```

```
val centralKategorieDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/central?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_kategorie")).load()
```

```
val allKategorieDF = centralKategorieDF.
  union(eastKategorieDF).
  union(westKategorieDF).
  dropDuplicates(Array("k_id"));
```

```
val eastDepartamentyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/east?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_departamenty")).load()
```

```
val westDepartamentyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/west?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_departamenty")).load()

val centralDepartamentyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/central?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_departamenty")).load()
```

```
val allDepartamentyDF = centralDepartamentyDF.
  union(eastDepartamentyDF).
  union(westDepartamentyDF).
  dropDuplicates(Array("d_id"));
```

```
val tempProduktyDF = allProduktyDF.
  join(allKategorieDF, allProduktyDF("p_k_id").
    equalTo(allKategorieDF("k_id")), "leftouter").
  join(allDepartamentyDF, allProduktyDF("p_d_id").
    equalTo(allDepartamentyDF("d_id")), "leftouter");
```

Prawie już jesteśmy w domu. Jeszcze tylko musimy zamienić identyfikator typu na jego nazwę. Aby to zrobić musimy wykorzystać własną funkcję w ramach *Dataset API*. Są na to dwa sposoby.

Dla przykładu zobacz:

<https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql-udfs.html>

Nasza funkcja nie ma być elementem standardowo dostępnym w Spark SQL dlatego moglibyśmy wykorzystać wariant poniższy, warto o tej możliwości pamiętać:

```
def getType = (typ_id: Integer) => {
  if (typ_id == 1) "Filmy"
  else if (typ_id == 2) "Gry"
  else "Zywnosc"
}

val getTypeUDF = udf(getType)

val produkty_DS = temp_produkty_DS.withColumn("typ", getTypeUDF($"p_t_id")).
  withColumn("id_produktu", $"p_id".cast("integer")).
  drop("p_k_id").drop("p_d_id").drop("p_t_id").
  drop("k_id").drop("d_id").drop("p_id").
  withColumnRenamed("p_nazwa", "nazwa_produktu").
  withColumnRenamed("k_nazwa", "nazwa_kategorii").
  withColumnRenamed("d_nazwa", "nazwa_departamentu")
```


My jednakże podejźmy do sprawy bardzo klasycznie, i chyba mniej problematycznie

```
val typyDF = Seq((1, "Filmy"), (2, "Gry"), (3, "Zywnosc")).toDF("t_id", "typ")

val produktyDF = tempProduktyDF.
  join(typyDF, tempProduktyDF("p_t_id").equalTo(typyDF("t_id")), "leftouter").
  withColumn("id_produktu", $"p_id".cast("integer")).
  drop("p_k_id").drop("p_d_id").drop("p_t_id").drop("k_id").
  drop("d_id").drop("p_id").drop("t_id").
  withColumnRenamed("p_nazwa", "nazwa_produktu").
  withColumnRenamed("k_nazwa", "nazwa_kategorii").
  withColumnRenamed("d_nazwa", "nazwa_departamentu")

produktyDF.write.insertInto("w_produkty");
```

3. `w_czas`: Czas na transformację dotyczącą **dat**. Do tej pory przetwarzanie XML traktowaliśmy trochę niepoważnie. Tym razem byśmy tego nie chcieli. Niestety domyślnie Spark nie posiada takiej funkcjonalności, dlatego potrzebowaliśmy skorzystać z dodatkowych bibliotek.

```
val dniWolneDF = spark.read.
  format("com.databricks.spark.xml").
  option("rootTag", "DNI_WOLNE").
  option("rowTag", "DATA").
  load(s"/user/$username/labs/spark/externaldata/dni_wolne.xml")
```

dniWolne_DS: org.apache.spark.sql.DataFrame = [DZIEN: string]

```
val eastDatyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/east?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_sprzedaz")).load().select("s_data");
```

```
val westDatyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/west?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_sprzedaz")).load().select("s_data");
```

```
val centralDatyDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/central?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_sprzedaz")).load().select("s_data");
```

```
val allDatyDF = centralDatyDF.
    union(eastDatyDF).
    union(westDatyDF).
    distinct();

val tempDatyDF = allDatyDF.
    join(dniWolneDF, allDatyDF("s_data").
        equalTo(dniWolneDF("DZIEN")), "leftouter");
```

O ile wcześniej postanowiliśmy nie korzystać z własnych funkcji wykorzystywanych przez *Spark SQL API*, o tyle teraz chyba od tego nie uciekniemy.

Musimy na podstawie dat wyznaczyć rok, miesiąc, kwartał oraz określić czy dany dzień był wolny czy nie – zamienić datę w kolumnie DZIEN na wartość true jeśli występuje i false w przeciwnym przypadku.

```
import org.apache.spark.sql.functions.udf
import spark.implicits._

val getYear: String => Integer = _.substring(0,4).toInt

val getMonth: String => Integer = _.substring(5,7).toInt

val isFree: String => Boolean = _ != null

val getYearUDF = udf(getYear)

val getMonthUDF = udf(getMonth)

val isFreeUDF = udf(isFree)
```

```
val datyDF = tempDatyDF.
    withColumn("rok", getYearUDF($"s_data")).
    withColumn("miesiac", getMonthUDF($"s_data")).
    withColumn("data", $"s_data".cast("date")).
    withColumn("czy_wolny", isFreeUDF($"DZIEN")).
    drop($"DZIEN").drop($"s_data")

datyDF.write.insertInto("w_czas");
```

4. **f_fakty**: Pozostała nam ostatnia stosunkowo nudna transformacja dotycząca tabeli **faktów**.
Wszystko co się w niej pojawi już było.

```
val eastSprzedazDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/east?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_sprzedaz")).load()

val westSprzedazDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/west?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_sprzedaz")).load()

val centralSprzedazDF = spark.read.format("jdbc").options(
  Map("url" -> s"jdbc:mysql://$ip_mysql:3306/central?autoReconnect=true",
    "driver" -> "com.mysql.jdbc.Driver",
    "user" -> "root",
    "password" -> pass_mysql,
    "dbtable" -> "etl_sprzedaz")).load()
```

```
val allSprzedazDF = centralSprzedazDF.
  union(eastSprzedazDF).
  union(westSprzedazDF);

val sprzedazDF = allSprzedazDF.
  withColumn("cena_sprzedazy", $"s_cena_sprzedazy".cast("double")).
  withColumn("cena_zakupu", $"s_cena_zakupu".cast("double")).
  withColumn("liczba_towarow", $"s_liczba_towarow".cast("integer")).
  withColumn("liczba_klientow", $"s_liczba_klientow".cast("integer")).
  withColumn("id_sklepu", $"s_sk_id".cast("integer")).
  withColumn("data", $"s_data".cast("date")).
  withColumn("id_produktu", $"s_p_id".cast("integer")).
  drop("s_data").drop("s_sk_id").drop("s_p_id").
  drop("s_cena_sprzedazy").drop("s_cena_zakupu").
  drop("s_liczba_towarow").drop("s_liczba_klientow");

sprzedazDF.write.insertInto("f_fakty");
```

Zadanie 3 – opracowanie analiz wyników projektu

1. Utwórz wynikowe analizy projektu
 - Analiza/porównanie sprzedaży w poszczególnych regionach w poszczególnych latach dla wybranych kategorii produktów
 - Analiza/porównanie zysku (sprzedaż minus koszt) ze sprzedaży poszczególnych kategorii w poszczególnych latach dla wybranych typów dni (wolnych/roboczych).

Wymagania, sugestie i podpowiedzi

- Ze względu na rozdzielone katalogi Apache Spark oraz Apache Hive w HDP 3.0 nie możesz skorzystać z *Data Analytics Studio*
- Możesz natomiast użyć notatnika Zeppelina, interpretera Sparka2 oraz API w Scali – w tym przypadku możesz skorzystać z graficznej wizualizacji wyników zapytań.

Jeśli uzupełnisz swoje notatniki o wykorzystanie możliwości jakie daje interpreter angulara, możesz z tego uzyskać notatniki, które nie tylko będą dobrze wyglądały, ale także będą interaktywne.

Zagłębij do dokumentacji Zeppelina

- http://zeppelin.apache.org/docs/0.8.0/usage/display_system/angular_backend.html
- http://zeppelin.apache.org/docs/0.8.0/usage/display_system/angular_frontend.html

- Transformacje pivot mogą znacząco przybliżyć wygląd uzyskanych wyników do oczekiwanych rezultatów (nie jest to wymagane – ważne aby w uzyskanych wynikach zapytań znalazły się dane wymagane przez analizy).

Rozwiązanie

Analiza 1

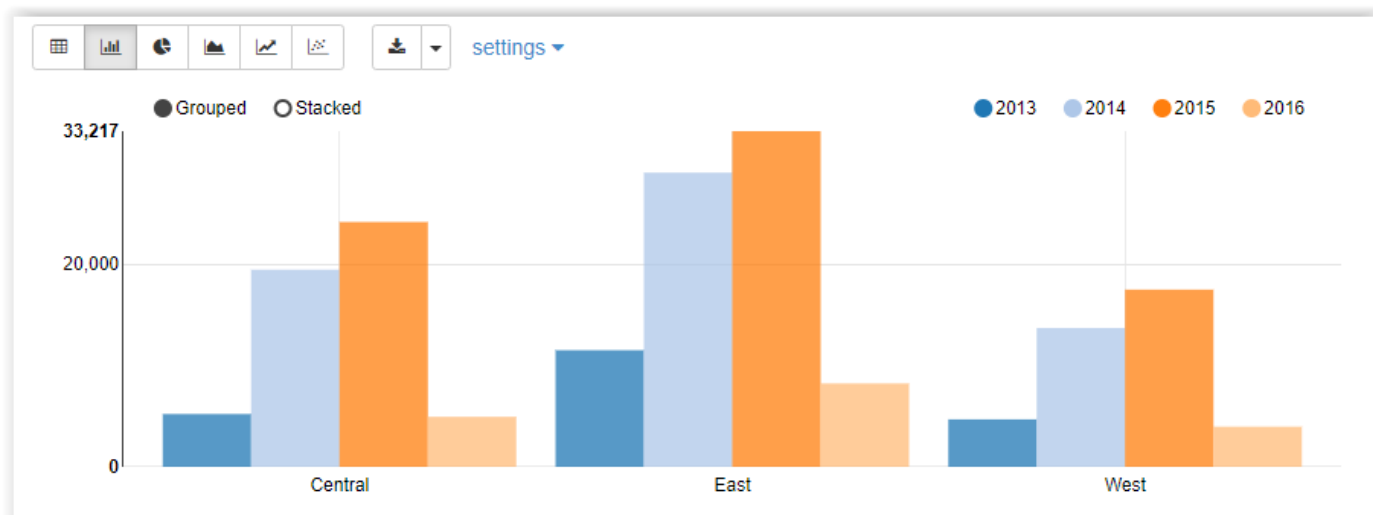
```
import org.apache.spark.sql.functions._
val sprzedazDF = spark.table("f_fakty").
  select("data", "id_sklepu", "id_produktu", "liczba_towarow", "cena_sprzedazy");
val sklepyDF = spark.table("w_sklepy").select("id_sklepu", "region");
val czasDF = spark.table("w_czas").select("data", "rok");
val produktyDF = spark.table("w_produkty").
  select("id_produktu", "nazwa_kategorii");

val analiza1DF = sprzedazDF.
  join(sklepyDF, sprzedazDF("id_sklepu") === sklepyDF("id_sklepu")).
  join(czasDF, sprzedazDF("data") === czasDF("data")).
  join(produktyDF, sprzedazDF("id_produktu") === produktyDF("id_produktu")).
  withColumn("wartosc_sprzedazy", $"liczba_towarow"*$"cena_sprzedazy").
  groupBy(sklepyDF("region"), produktyDF("nazwa_kategorii")).
  pivot("rok").
  agg(round(sum("wartosc_sprzedazy"),2).as("suma_sprzedazy"))
```

```
val sci-fi = analiza1DF.
  where($"nazwa_kategorii" === "Sci-Fi").drop("nazwa_kategorii").
  orderBy($"region".asc)

sci-fi.createOrReplaceTempView("analiza1_sci-fi");
```

```
%sql
select * from analiza1_sci-fi
```



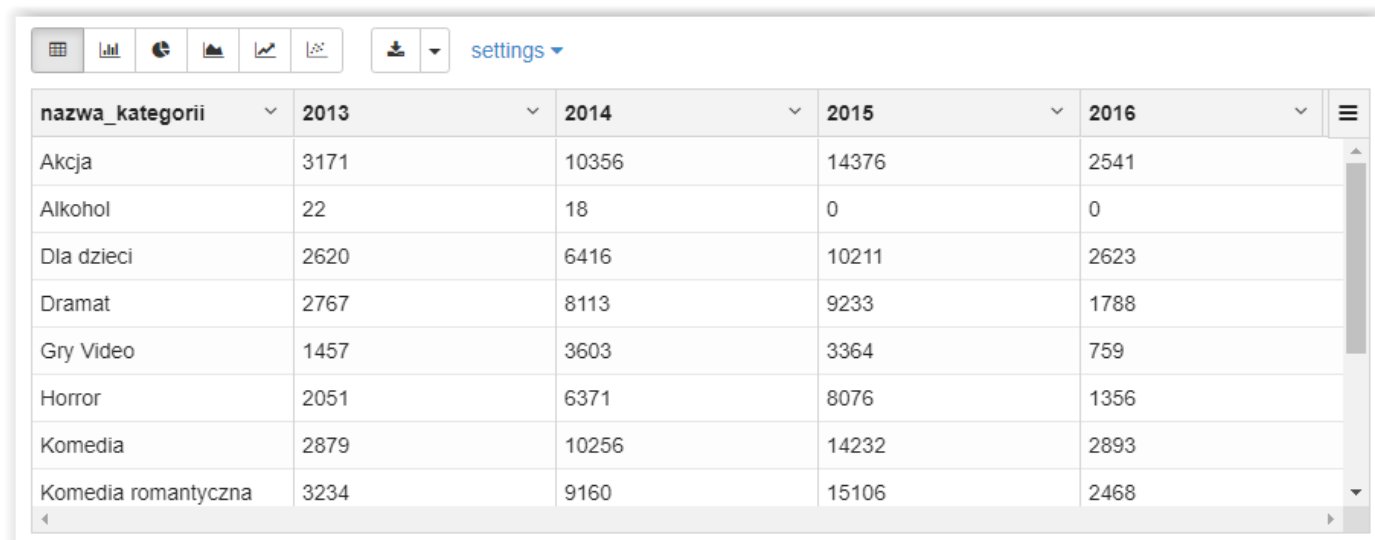
Analiza 2

```
val sprzedazDF = spark.table("f_fakty").
  select("data", "id_produktu", "liczba_towarow", "cena_zakupu", "cena_sprzedazy");
val czasDF = spark.table("w_czas").select("data", "rok", "czy_wolny");
val produktyDF = spark.table("w_produkty").
  select("id_produktu", "nazwa_kategorii");

val analiza2DF = sprzedazDF.
  join(czasDF, sprzedazDF("data") === czasDF("data")).
  join(produktyDF, sprzedazDF("id_produktu") === produktyDF("id_produktu")).
  withColumn("zysk", $"liczba_towarow" * ($"cena_sprzedazy" - $"cena_zakupu")).
  groupBy(produktyDF("nazwa_kategorii"), czasDF("czy_wolny")).
  pivot("rok").
  agg(round(sum("zysk"), 2).as("suma_zysku"))

val wolneDF = analiza2DF.where($"czy_wolny" === true).
  drop("czy_wolny").orderBy($"nazwa_kategorii".asc)
wolneDF.createOrReplaceTempView("analiza2_wolne");
```

```
%sql
select * from analiza2_wolne
```



nazwa_kategorii	2013	2014	2015	2016
Akcja	3171	10356	14376	2541
Alkohol	22	18	0	0
Dla dzieci	2620	6416	10211	2623
Dramat	2767	8113	9233	1788
Gry Video	1457	3603	3364	759
Horror	2051	6371	8076	1356
Komedia	2879	10256	14232	2893
Komedia romantyczna	3234	9160	15106	2468

Uważne osoby zapewne zauważyły, że nasze analizy mogłyby mieć postać po prostu zapytań SQL. Odwoływanie się do tych tabel składnią funkcyjną rozpoczynaną od `spark.table("nazwa_tabeli")` nie jest zbyt przyjazne. To samo dotyczy wielu innych miejsc w naszym projekcie np. transformacji ETL. Powodem tak zaprezentowanego kodu była chęć pokazania klasycznych metod przetwarzania obiektów `DataFrame` i `Dataset`. Ale czy tylko o to chodziło?

No właśnie... błędy dotyczące typów danych w typowanych operacjach da się wykryć na etapie kompilacji. Błędy w nietypowanych (np. `spark.sql`) wyjdą na etapie wykonania... To ma znaczenie, kiedy piszemy poważne fragmenty kodu, które następnie mają być wykonywane w poważnych środowiskach produkcyjnych.

Zakończenie

Na zakończenie zestawu zadań usuń oczywiście klaster.

Jeśli nie zamierzasz w najbliższym czasie korzystać z tej bazy danych MySQL, usuń koniecznie także instancję maszyny, która ją obsługuje.

