Data Engineering Exercise

Musixmatch

Job applicant: Fabio Stivanin

Index

[1 Introduction 3](#_Toc78967825)

[2 Tools 4](#_Toc78967826)

[3 Data preparation 5](#_Toc78967827)

[3.1 Sample data 5](#_Toc78967828)

[3.2 New tables 6](#_Toc78967829)

[4 Spark code on Databricks 9](#_Toc78967830)

[4.1 Environment set-up 9](#_Toc78967831)

[4.2 Task 1 – Count daily active devices 10](#_Toc78967832)

[4.3 Task 2 – Count daily active users 12](#_Toc78967833)

[4.4 Task 3 – Most viewed daily lyrics 13](#_Toc78967834)

[4.5 Task 4 – 50% Daily lyrics account 14](#_Toc78967835)

Introduction

The goal of the exercise is to test the following skills of the candidate:

* optimizing the underlying data
* creating one or more datasets to support visualizations tools.

The sample data is stored on a publicly exposed MySQL database. The database machine is intentionally underpowered, to simulate data optimization and preparation.

The sample data consists of two tables: user and views.

Each row on the table user represents a user profile, while each row on the table views represents a single lyrics visualization.

**Step 1:**

Prepare the data to lower the cardinality of the tables, if needed for speeding up the next step. You can either create new tables with just the field needed, correctly indexed; or modify the existing ones.

**Step 2:**

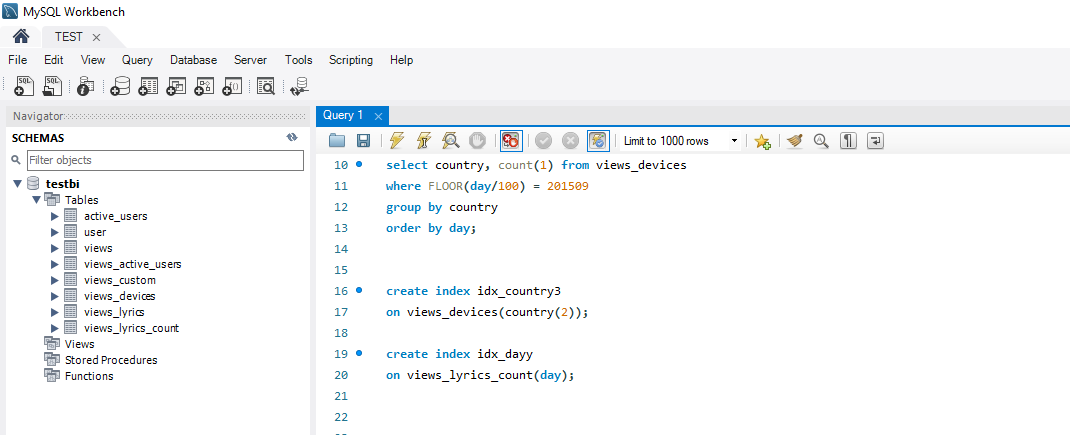
Using PySpark create one or more datasets to support a visualization tool that have to answer to the following questions:

1. How many devices are active daily? What product? In which countries? With what applications?
2. How many users are active daily? What product? In which countries? With what applications?
3. What are the most viewed daily lyrics by country and / or application?
4. How many distinct lyrics account daily, weekly and monthly for 50% of the views? Drill down by country and application. Is this constant over the full period of time

Tools

**MySQL Workbench**

To visualize tables’ structure and sample data, it was used MySQL Workbench ( [Download](https://www.mysql.com/it/products/workbench/) ), that is a visual database design tool that integrates SQL development, administration, database design, creation and maintenance into a single integrated development environment for the MySQL database system.



**Databricks**

To create the spark code it was used “Databricks community Edition” ( [Login - Databricks Community Edition](https://community.cloud.databricks.com/login.html) ) that is a free version of a cloud-based big data platform; a free account permits the use of a small Spark cluster.



Data preparation

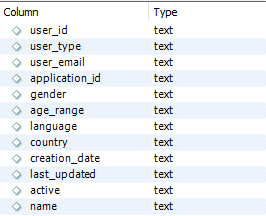
## Sample data

**Database connection parameters (MySQL v 5.6.10)**

url = "jdbc: mysql://mxm-testbi.c72srgqwk8ib.us-east-1.rds.amazonaws.com:3306/testbi"

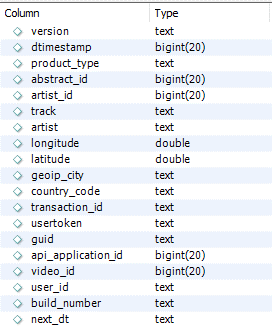
user= “mxmtest” password= <check mail>

**Table tesbi.USER**



The approximate number of rows is 41 million.

**Table testbi.VIEWS**



The approximate number of rows is 31 million, across a 1 year timespan.

## New tables

**Table VIEWS\_CUSTOM**

create table testbi.views\_custom as (

select trim(guid) as device

, trim(country\_code) as country

, CAST(from\_unixtime(dtimestamp, '%Y%m%d') as UNSIGNED) as DAY

, trim(product\_type) as product

, cast(api\_application\_id as UNSIGNED) as application

, trim(user\_id) as user

from views

);

First, we create a new table with only the interesting columns to optimize access and calculations on the table.

The trim() function removes blank spaces on the left and the right of the field.

Dtimestamp is transformed to a smarter format, easier to read and sort. The function unixtime() convert the timestamp date to the format YYYYMMDD (for example 20150901 stands for 1st September 2015)

The access on number (integer) columns is faster so we cast() day and application as unsigned number.

create index idx\_device

on views\_custom(device(136));

create index idx\_user\_id2

on views\_custom(user(36));

create index idx\_day

on views\_custom(DAY);

The relevant columns are indexed to enhance performance during the joins on views String columns need to specify the length of the field (it is set as the max(length()) of the field in the table).

**Table VIEWS\_DEVICES**

create table testbi.views\_devices as (

select device, country, day, product, application

from views\_custom

where device is not null and device <> '-'

);

The 1st question is focused on a count on the active devices. We can discard rows from table views where guid is not specified.

The cardinality of the table moves from the initial 31M rows to 15M (almost 50%).

The relevant columns are indexed.

**Table ACTIVE\_USERS**

To answer the 2nd question we only need to extract distinct active users.

create table active\_users as (

select distinct trim(user\_id)

from user

where active ='1'

);

The column was indexed to enhance performance during the joins on views. User\_id is a string, so we need to specify the length of the field in the statement.

The relevant columns are indexed.

The number of rows is now around 8M (20% of rows vs. the initial 41M).

**Table VIEWS\_ACTIVE\_USERS**

The join between views and active users is relatively slow so it is better to do it just one time and create a new table with only the views of active users. It is an inner join because we need an existing mapping between active users and views.

create table views\_active\_users as (

select distinct DAY, user, application, country, product

from views\_custom as a

join active\_users as b

on a.user=b.user\_id

);

The relevant columns of interest are indexed.

The number of rows is now around 700K, good performance for group by and count operations on this table.

**Table VIEW\_LYRICS**

create table testbi.views\_lyrics as (

select CAST(from\_unixtime(dtimestamp, '%Y%m%d') as UNSIGNED) as day

, cast(api\_application\_id as UNSIGNED) as application

, trim(country\_code) as country

, cast(abstract\_id as unsigned) as abstract\_id

from views

where trim(product\_type)= 'lyrics'

);

The 3rd question s based on lyrics so we can extract in a new table only views where product\_type= ‘lyrics’ (we could consider ‘lyrics-restricted’ too).

The new table has 10M rows (25% vs. initial table).

The relevant columns are indexed.

**Table VIEW\_LYRICS\_COUNT**

create table views\_lyrics\_count as (

select day, country, application, abstarct\_id

, count (1) as count\_views

from views\_lyrics

group by day, country, application, abstarct\_id

);

The daily views count is performed one-time and the results are stored in this new table. The cardinality is 8M rows.

To answer the 3rd question now we can sum the values of the column count\_views adapting the group by clause on the specific calculation.

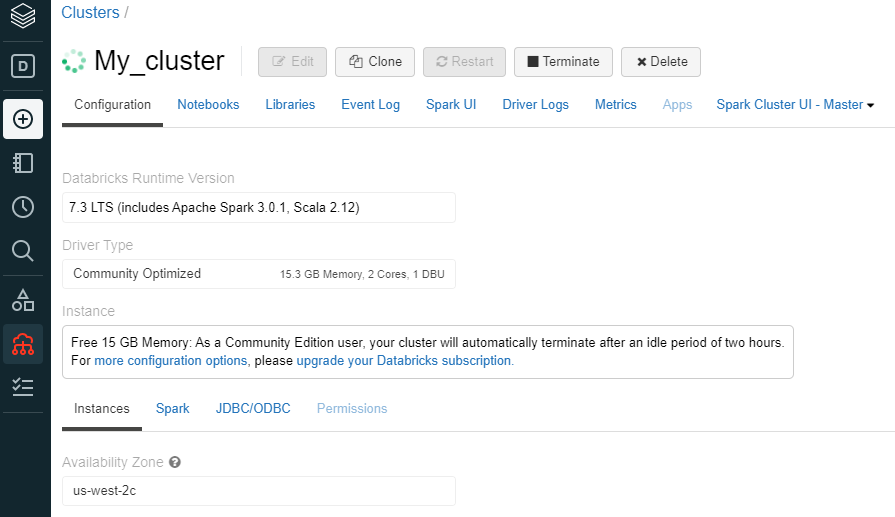
The relevant columns are indexed.

Spark code on Databricks

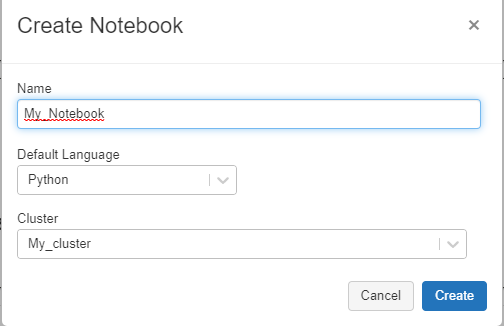
## Environment set-up

First of all, we need to connect to Databricks site and login. The community edition trial account let me access Databricks Notebooks and create a small cluster (<https://community.cloud.databricks.com/)>

On the left side bar click on *Compute* and then *Create Cluster*.



Then, create a new Notebook attached to the previous created Cluster. Set the default language to python.



Connect to JDBC database

%python

driver = "org.mariadb.jdbc.Driver"

url = "jdbc:mysql://mxm-testbi.c72srgqwk8ib.us-east-1.rds.amazonaws.com:3306/testbi"

user = "mxmtest"

password = "XXX"

## Task 1 – Count daily active devices

**How many devices are active daily? What product? In which countries? With what applications?**

First, let's create a DataFrame for table views\_devices.

To query this data as a table, it is simple to register it as a view *(dataFrame.createOrReplaceTempView("X"))* or a table *(dataFrame.write.saveAsTable(“X”)).*

%python

table= "views\_devices"

device\_table = spark.read.format("jdbc")\

  .option("driver", driver)\

  .option("url", url)\

  .option("dbtable", table)\

  .option("user", user)\

  .option("password", password)\

  .load()

device\_table.createOrReplaceTempView("DEVICES") # view

# device\_table.write.saveAsTable("DEVICES\_TABLE") # table

Now we can investigate the DEVICE data using queries like the following one that count the devices that are active on a specific day, group by country, application and product.

We can limit the results to the 10 results with the biggest count.

%sql

select count(distinct device) as count\_devices

, a.day

  from DEVICES a

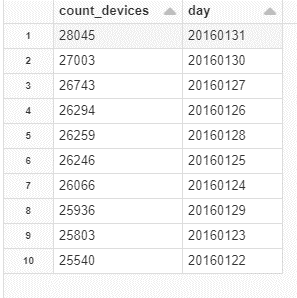
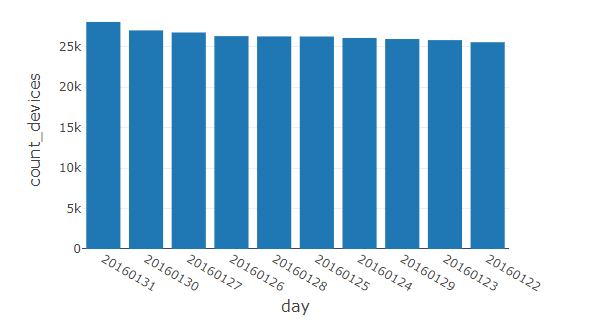
where floor(day/100) = 201601

  group by a.day

  order by count\_devices desc

  limit 10

Databricks editor allows you to easily visualize query results as a grid or a plot.



For example, in the month of January 2016 the day with the biggest count of active devices is the 31st.

In the plot and grid we can see the top 10 days.

If we focus, for example, on 2016 Jan 31st and group by other columns, the result is the following.

%sql

select count(distinct device) as count\_devices

, a.day, product, country, application

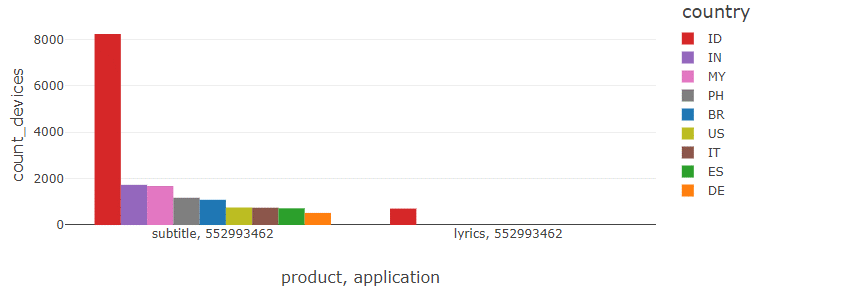
 from DEVICES a

where day= 20160131

  group by a.day, product, country, application

  order by count\_devices desc

  limit 10



The most of devices come from the country Indonesia. Subtitle is the most viewed product with application id 552993462. In other countries the number of active devices is far lower that day.

## Task 2 – Count daily active users

**How many users are active daily? What product? In which countries? With what applications?**

%python

table= "views\_active\_users"

user\_table = spark.read.format("jdbc")\

  .option("driver", driver)\

  .option("url", url)\

  .option("dbtable", table)\

  .option("user", user)\

  .option("password", password)\

  .load()

user\_table.createOrReplaceTempView("USERS")

Here is the query.

%sql

select count(distinct user) as count\_users

, a.country

, a.application

, a.product

, a.day

  from USERS a

  group by a.country

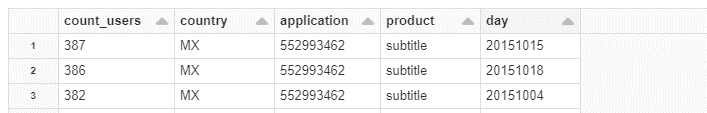
, a.application

, a.product

, a.day

  order by count\_users desc

Thank to previous aggregation on data, count operations on daily active users are quite fast.



The day with the biggest number of active user is 2015 Oct 15th, from Mexico, subtitles and application id 552993462.

## Task 3 – Most viewed daily lyrics

**What are the most viewed daily lyrics by country and / or application?**

%python

table= "views\_lyrics\_count"

lyrics\_table = spark.read.format("jdbc")\

  .option("driver", driver)\

  .option("url", url)\

  .option("dbtable", table)\

  .option("user", user)\

  .option("password", password)\

  .load()

lyrics\_table.createOrReplaceTempView("LYRICS")

Here we just need to sum the views, that is a faster aggregation than a count(distinct).

We used the function rank() over (partition by … order by …) to extract the top daily lyrics.

%sql

select rank() over (partition by application, country  order by sum(count\_views) desc ) as ranking

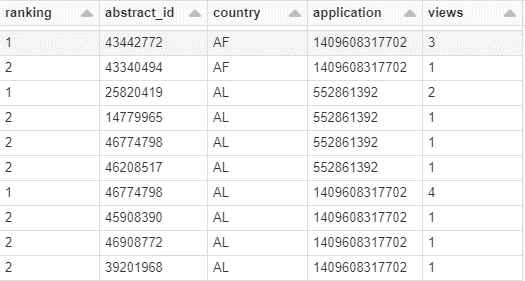
, country, application, abstract\_id

, sum(count\_views) as views

from LYRICS a

where day = 20160101

group by abstract\_id, country, application



## Task 4 – 50% Daily lyrics account

**How many distinct lyrics account daily, weekly and monthly for 50% of the views? Drill down by country and application. Is this constant over the full period of time?**

The following query returns the min number of distinct lyrics needed to reach the 50% of daily/weekly/monthly total views.

We need to calculate in different subqueries (using with clause):

* The ranking of the lyrics based on the number of views in the period 🡪 we use row\_number() function here instead of rank() because we need to assigns a unique, sequential number to each row
* The total number of views in the period
* The cumulative views based on the lyrics ranking (we are interested on the most viewed lyrics)
* The percentage vs. total
* The min(ranking), that is the min number of distinct lyrics, such that the percentage is more than or equal to 50%

%sql

with ranking as (

SELECT abstract\_id, sum(count\_views) as views

,row\_number() over (order by sum(count\_views) desc) as ranking

FROM LYRICS

where

day=20160101 --daily

--day between 20160101 and 20160107 --weekly

--floor(day/100) = 201601 --monthly

group by abstract\_id

)

, total as (

SELECT sum(count\_views) as tot\_views

FROM LYRICS

where

day=20160101 --daily

--day between 20160101 and 20160107 --weekly

--floor(day/100) = 201601 --monthly

)

, cumulative\_views as (

select abstract\_id, views, ranking

, SUM(views)

          over ( ORDER BY ranking ROWS BETWEEN unbounded preceding AND CURRENT ROW ) as cumsum

from ranking

order by cumsum

)

, percentage as (

select a.abstract\_id, views, ranking, cumsum, tot\_views

, cumsum/tot\_views \* 100 as perc

from cumulative\_views a

, total b

)

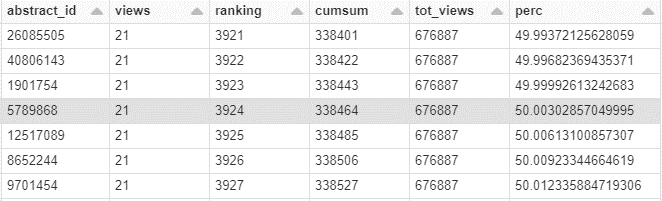
select min(ranking) as count\_fifty\_perc\_views

from percentage

where perc >= 50

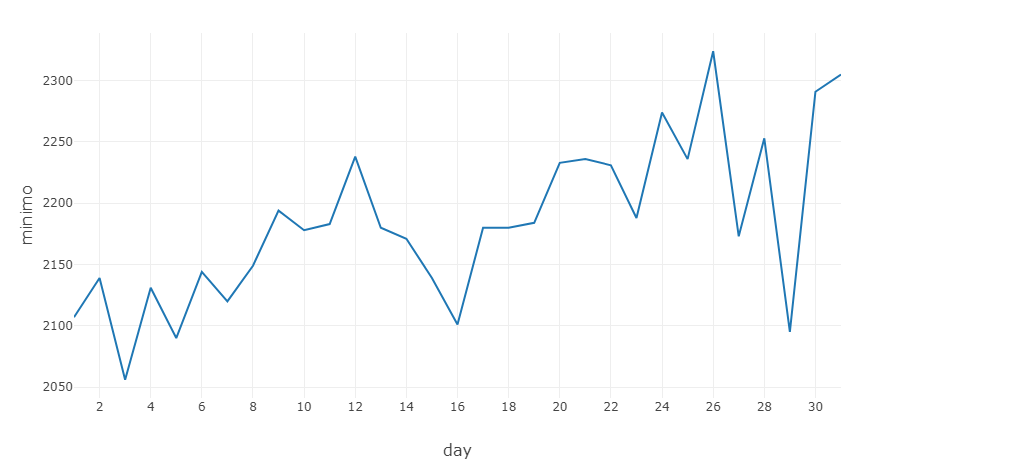
For example, considering the month of January 2016, the total number of views is 676.887.

The min number of distinct lyrics that account for 50% of monthly views is 3.924, as we can see from the following grid.

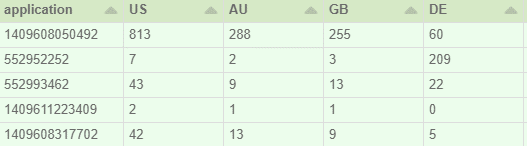


If we add the column “day” to the group by and partition by clauses in the query, we can see the daily trend in the period.

The following is the daily trend for January 2016. We can see some peaks at day 12and 26.



If we also add the column “country” and “application” to the group by and partition by clauses, we can drill down by country and application. Useful for pivot tables, like the following.



Focus on 2016 January 1st.

