



**Recomendador Musical**

**MEMORIA**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Proyecto:** | Proyecto Fin de Master | | |  |  |  |
| **Autores:** | Carlos López Pita  Jesús de Lucas Rodríguez  Javier Bermejo Izquierdo | | |  | **Código:** | RecomendadorMaster |
| **Versión:** | 1 |  |  |  | **Código interno:** | N/A |
| **Verificado:** | Carlos López Pita  Jesús de Lucas Rodríguez  Javier Bermejo Izquierdo | | |  | **Autorizado**: | N/A |
| **Modificación:** | 20/03/2017 | | |  |  |  |
|  | | | |  |  | |
| C/ Manuel Tovar 49-53  28034 Madrid  Tel: +34 917 680 490 / Fax: +34 913 833 301 | | | | | | |
| Reservados todos los derechos.  © Pragsis - Bidoop, 2015. | | | | | | |

# Control de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| Versión | Fecha | Autor | Descripción |
| Versión 1 | 20/03/2017 | Carlos López Pita  Jesús de Lucas Rodríguez  Javier Bermejo Izquierdo | Versión inicial del documento |

# Contenido

Control de versiones 2

Descripción del documento 4

Requisitos 5

Esquema general 6

Procesos en batch 7

Procesos en streaming 13

# Descripción del documento

Este documento es la memoria del ejercicio práctico propuesto para el fin de master de Big Data organizado por Pragsis.

El ejercicio consiste en el diseño, implementación e implantación de un sistema de recomendación musical y envío de publicidad para una plataforma de música online.

# Requisitos

Se partirá del volumen actual de usuarios que tiene la compañía en una base de datos Oracle y de un sistema de eventos generados por los usuarios en el momento en el que reproducen una canción.

A partir de estos datos nuestro trabajo consiste en:

* Recibir los eventos del sistema de mensajería proporcionado por la empresa, en nuestro caso, a través de un servidor de eventos JMS.
* Recomendar a cada usuario un grupo cada vez que escuche una canción a través de un sistema de recomendación basado en las preferencias que han generado otros usuarios al escuchar sus canciones.
* Envío de publicidad a través de un banner cuando el usuario escuche X canciones, según unos parámetros especificados por el cliente.
* Registrar un histórico de todas las canciones escuchadas por los usuarios durante 5 años.
* El sistema tiene que estar protegido contra fallos cuando lleguen los eventos y poder ser escalable tanto para nuevos usuarios como para nuevos eventos generados.

# Esquema general

La solución aportada tras la presentación con el cliente va a constar de dos grandes procesos en los que se enfocarán los requerimientos exigidos por la empresa para el sistema de recomendación.

* Procesos Batch:

1. Volcado de los usuarios actuales y futuros de la plataforma de música a partir de la base de datos Oracle a HDFS.
2. Calcular el modelo para que pueda ser utilizado al hacer las recomendaciones.
3. Gestionar los banners de publicidad.
4. Crear y actualizar las tablas HIVE necesarias para los procesos en streaming.

* Procesos en Streaming:

1. Recepción de los eventos y gestionarlos a los demás procesos.
2. Almacenamiento en HDFS de los mismos.
3. Generar la recomendación para cada usuario
4. Generar también el banner de publicidad cuando cada usuario escuche X canciones.

Tanto los usuarios como los eventos serán almacenados en HDFS en el directorio /user/cloudera/mrec/, donde dentro del mismo habrá un directorio users, otro events y otro banners.

# Procesos en batch

## Almacenamiento de usuarios

Para la gestión de los usuarios vamos a utilizar Sqoop.

Primeramente, vamos a volcar el histórico de usuarios que tiene de momento el cliente en una base de datos Oracle a HDFS, para ello se ha creado el directorio /user/cloudera/mrec/users/ donde se irán almacenando los usuarios por fecha. Para nuevas altas se irán guardando en HDFS en el directorio correspondiente a la fecha de alta.

Los usuarios van a estar ordenados por su ID, para ello vamos a transformar el String que nos viene por un Integer, eliminando la subcadena “user”. Con este ID vamos a poder gestionar las nuevas altas de usuarios gracias al --incremental append de sqoop que nos permitirá ir guardando en HDFS solo los nuevos usuarios.

El job de sqoop será lanzado con Ozzie cada noche para guardar los nuevos usuarios.

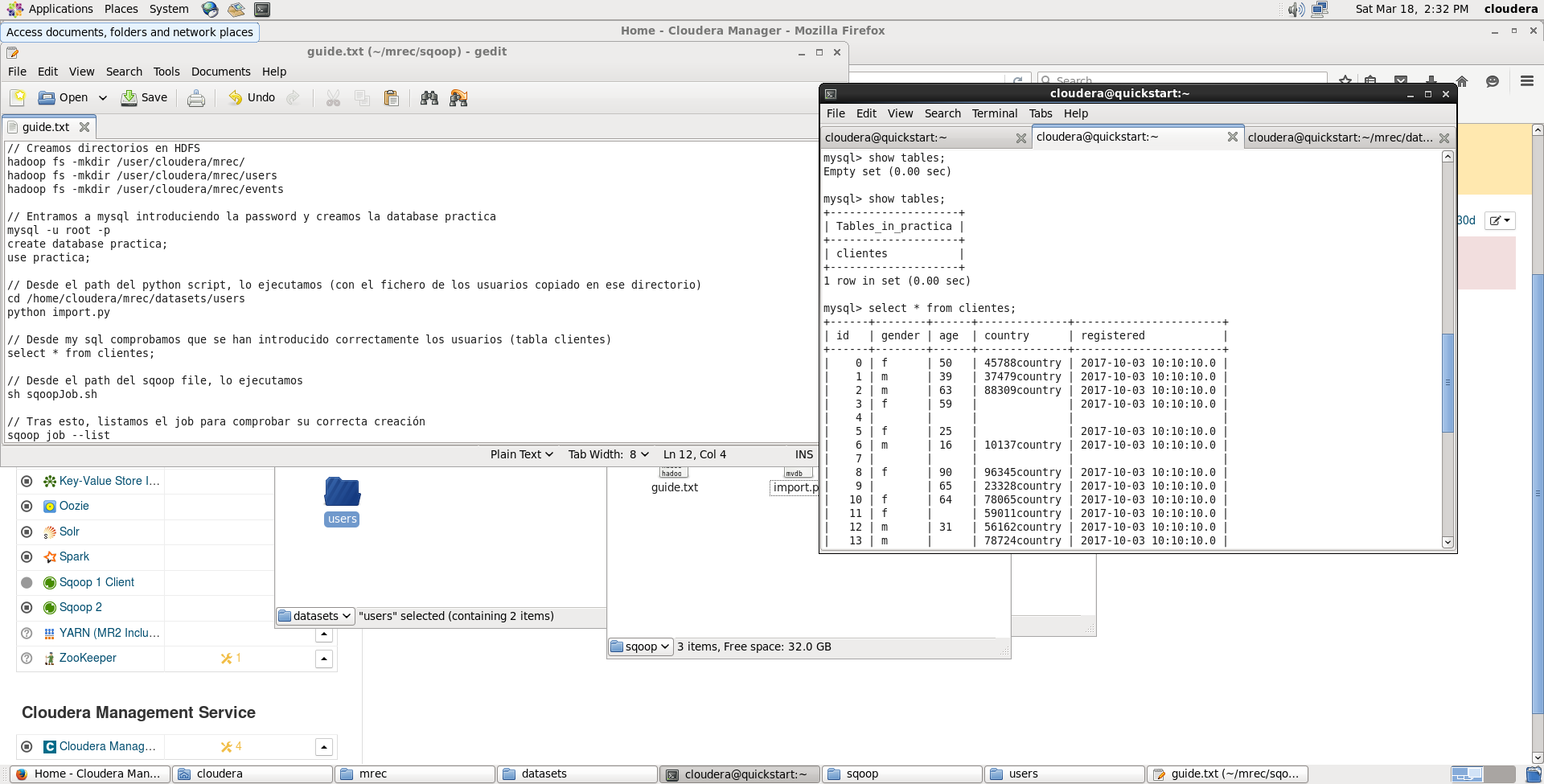


Tabla 1.1 - Visualización de los usuarios en mysql

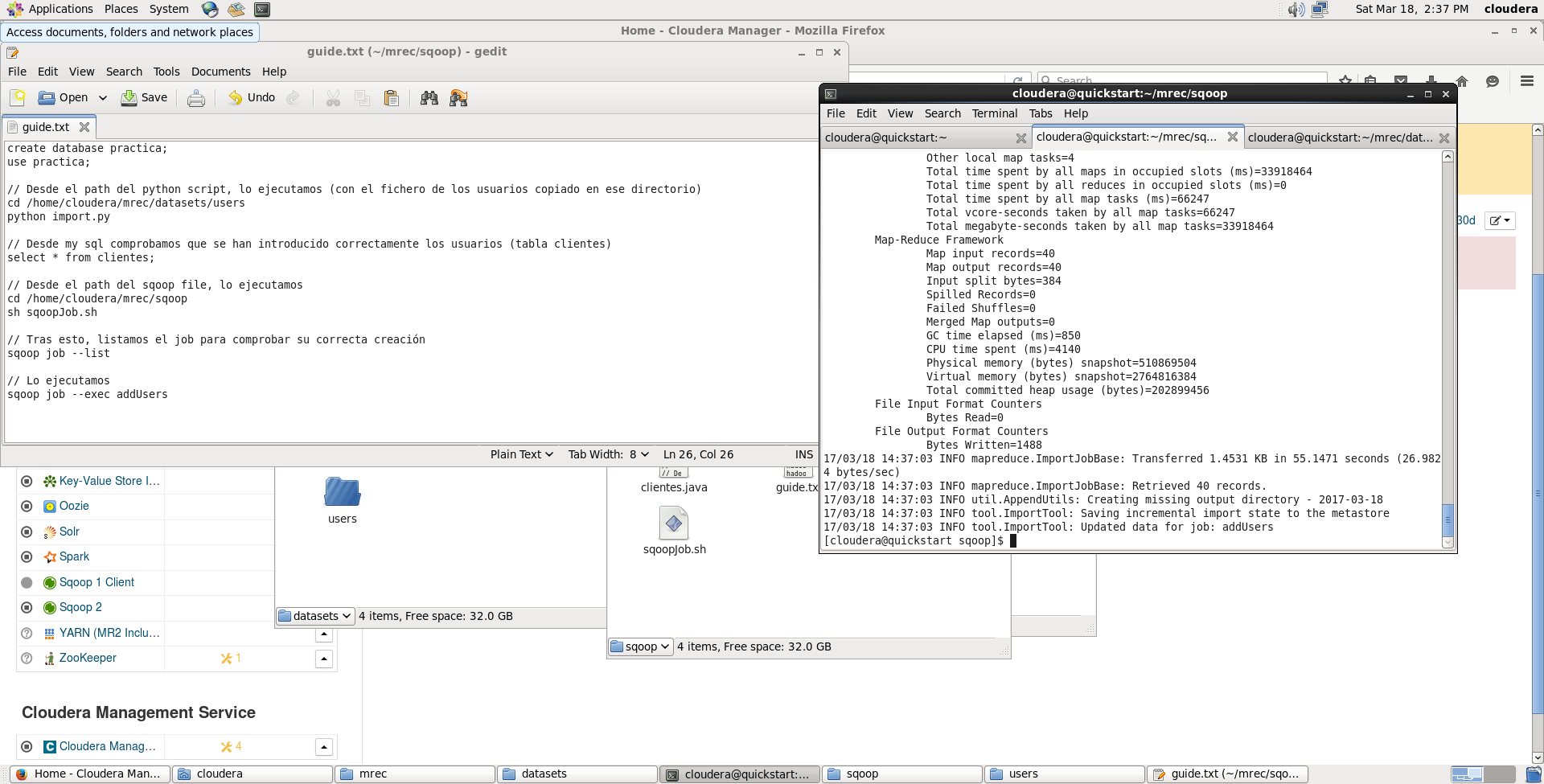


Tabla 1.2 - Volcado de los usuarios a HDFS

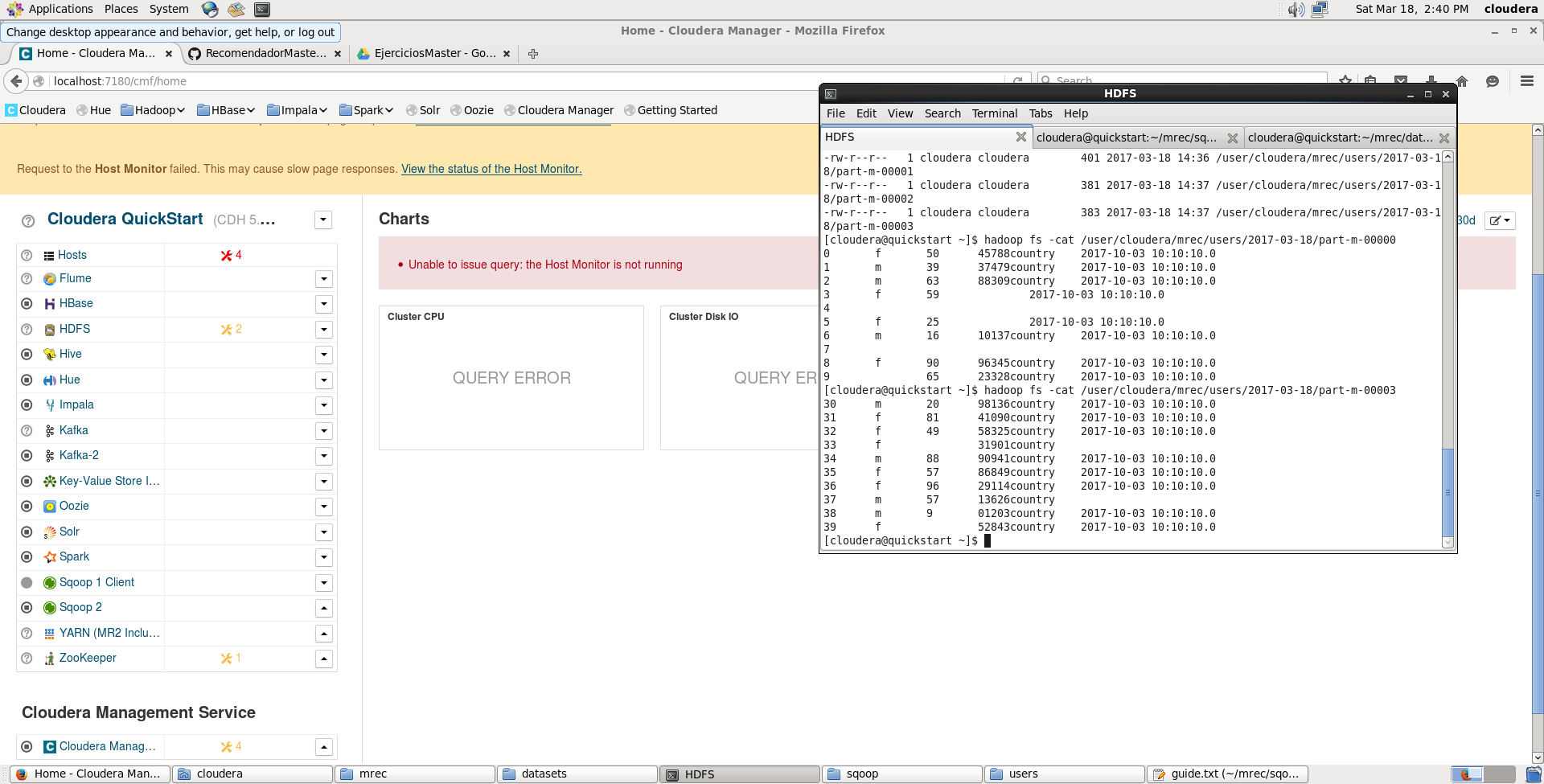


Tabla 1.3 - Listado de los usuarios en HDFS

## Creación y población de tablas HIVE para los procesos.

Vamos a crear una serie de tablas en HIVE para poder usar en el proyecto. Para ello vamos a utilizar como clave el userID y estarán particionadas por la fecha.

Las tablas son las siguientes:

* Clientes: tendrá toda la información de los usuarios de la plataforma.
* Eventos: tendrá la información relativa de los eventos que han llegado al sistema.
* Numrepuserartist: número de reproducciones de un usuario.
* artidWithNum: artista con un integer que lo identifica.
* ratingsperartist: ratings por artista.
* ratingsWithArtist: ratings de usuario por artistas.

La tabla de clientes la obtenemos del apartado anterior. El resto de tablas las obtendremos de HDFS una vez se hayan obtenido por streaming los eventos.

Para crear las tablas ejecutamos el fichero create\_tables con HIVE. Luego añadimos location y las particiones con el fichero add\_partitions con HIVE. Y creamos las tablas de ratings con el fichero create\_ratings con HIVE también.

## Banners de publicidad.

La gestión de los banners consiste en que el cliente nos proporcionará la información sobre qué tipos de banners quiere que se les muestre a los usuarios según su edad y sexo. A partir de esa información se generará una tabla que se almacenará en HDFS en el directorio /user/cloudera/mrec/banners/. Esta tabla podrá ser modificada según las necesidades del cliente en cada momento.

A partir de ahora lo que hacemos es leer los usuarios y los banners de HDFS y los cargamos en tablas de HBASE. Para ello vamos a utilizar Spark. El proceso es el siguiente:

* Cargamos los usuarios de HDFS a HBASE.
* Cargamos los banners de HDFS a HBASE.
* Añadimos a la tabla de usuarios de HBASE otra columna donde aparecerá el banner que se le ha asignado a cada usuario a partir de la tabla de banners.

Con todo esto, tendremos guardados los usuarios y su banner en HBASE al que se podrá acceder posteriormente en el proceso streaming para devolver la publicidad cuando el usuario reproduzca X canciones.

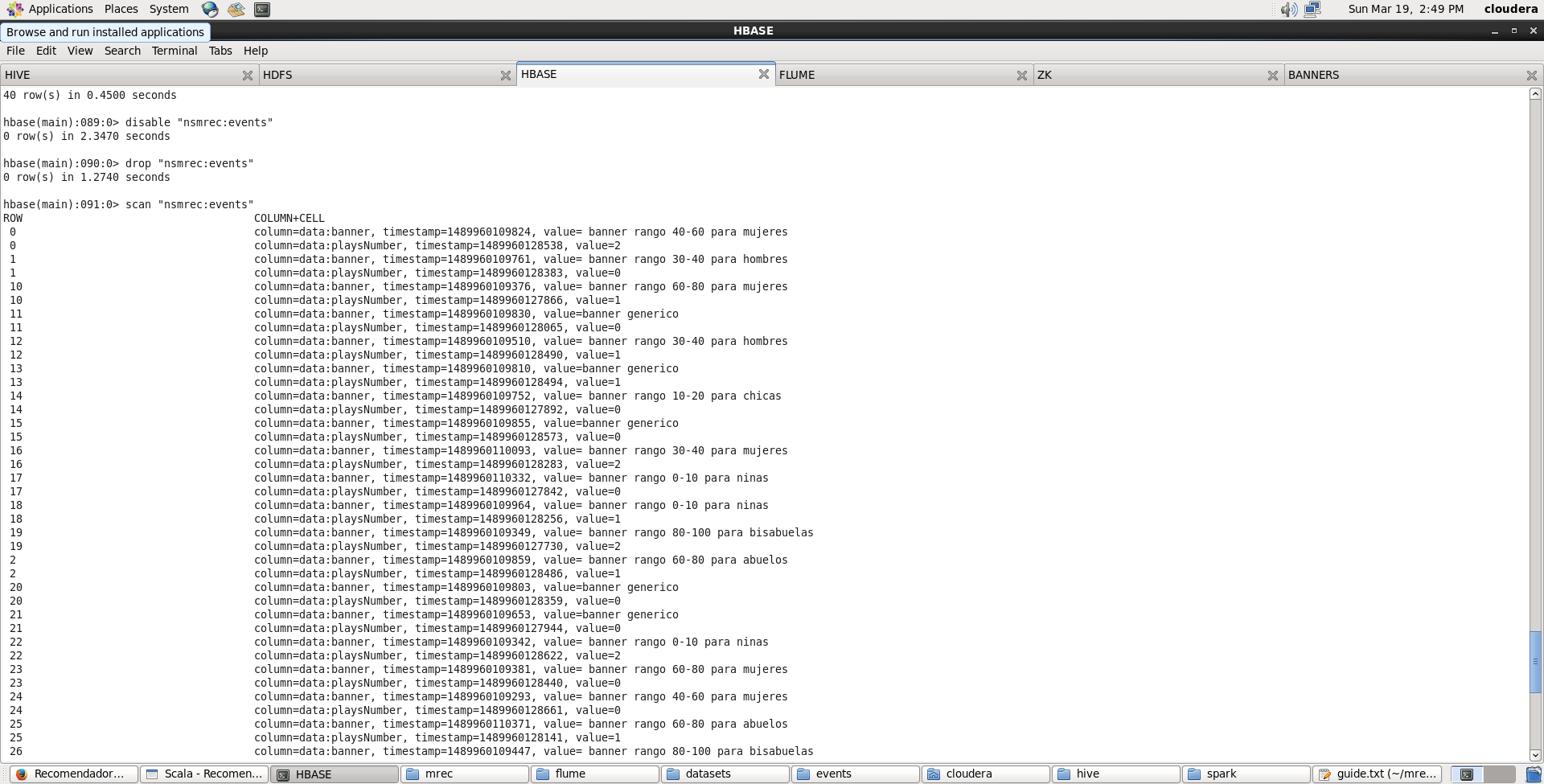


Tabla 2.1 - Tabla HBASE con banners

## Recomendador musical (Batch)

Para el recomendador musical vamos a utilizar el algoritmo ALS con el que vamos a generar un modelo a través de la información histórica de eventos que nos han proporcionado y con el cual podremos hacer predicciones en streaming de los artistas que pueden gustar a los usuarios.

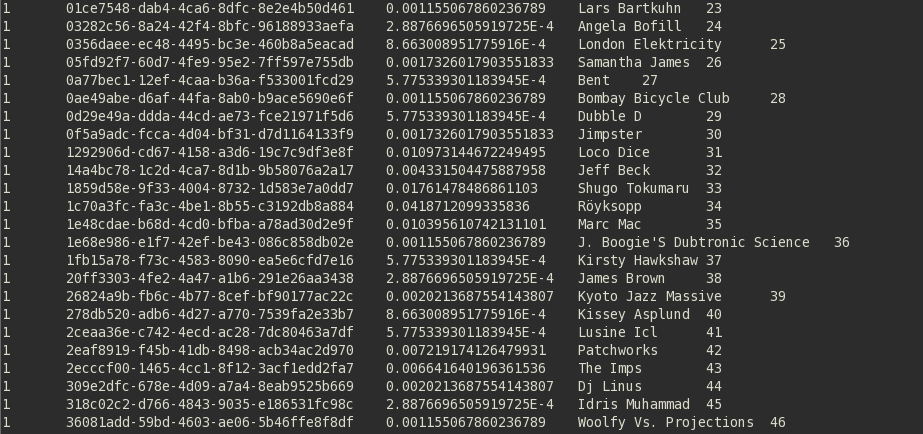
Con SPARK vamos ejecutar la aplicación crearModelo.scala en la que vamos a utilizar un contexto de HIVE y hacer una consulta sobre la tabla ratingsWithArtist que generamos en el apartado 2 y contiene los ratings de los usuarios con cada artista. 

Tabla 3.1 - Tabla con los ratings que se usará para la creación del modelo

SPARK en sus últimas versiones nos proporciona la función als.fit() con la que podremos entrenar el modelo usando el DataFrame obtenido a partir de la tabla de ratings y lo guardaremos en la ruta seleccionada.

Vamos a realimentar la tabla ratingsWithArtist y ejecutar la aplicación crearModelo.scala cada noche a través de Ozzie para poder tener cada vez un modelo más preciso para realizar las recomendaciones.

Una vez entrenado el modelo ejecutamos el CargaDirectArtist.scala que nos generará los ficheros en hdfs con los ids de los artistas junto con sus nombres. Este fichero se usará posteriormente en otro programa para cargar los artistas en Hbase para hacer las recomendaciones.

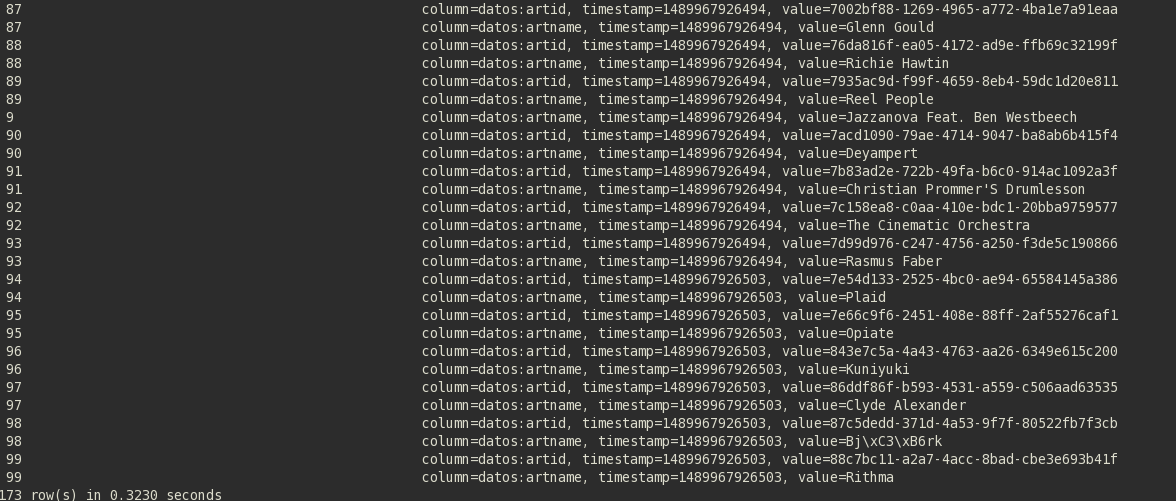


Tabla 3.2 - Tabla: artistWithName que contiene el artid y su nombre

Este fichero es el que usará el programa: CargaArtistHBase.scala para generar la tabla: artistWithName

# Procesos en streaming

## Recepción y gestión de los eventos.

Los eventos vendrán a través de un servidor JMS, para el proyecto utilizamos el fichero .tsv de eventos proporcionado que contendrá los siguientes campos <user, timestamp, artist, song>.

Lo primero que hacemos es crear un topic de Kafka que va a recibir los eventos y derivar a los distintos consumidores que necesitaremos para hacer las recomendaciones de artistas y publicidad, así como guardarlos en HDFS. Una vez comprobado que se crea el topic correctamente realizamos las siguientes operaciones:

Mediante Apache FLUME nos creamos un agente, en el cual vamos a utilizar como source un spooldir para leer del contenido del directorio el fichero .tsv con los eventos, en el caso del servidor JMS pues el source sería un jms.

Como channel ponemos un kafkaChannel que es por donde queremos ir almacenando y gestionando los eventos para luego repartir a los distintos consumidores.

Y como sink ponemos un HDFS Sink donde queremos escribir los eventos para su posterior utilización.

Una vez lanzado el agente FLUME también lanzamos los consumidores de Kafka para los otros procesos de streaming, en nuestro caso el consumidor de banners y el consumidor de usuarios para recomendar.

## Almacenamiento de eventos en HDFS.

Los eventos serán almacenados en HDFS según la fecha en la que lleguen en el directorio /user/cloudera/mrec/events/%Y-%m-%d a través de KAFKA. Son almacenados como DataStream y serán utilizados por los procesos Batch para actualizar el modelo de recomendaciones.

## Banners publicitarios.

Una vez que el cliente nos ha dado las condiciones para generar los banners, estos se han tenido que haber generado mediante un proceso batch donde se le indicaría a cada usuario el banner que se va a mostrar. Para ello se tuvo que ejecutar la aplicación spark GenerarBannerHDFS.scala citada anteriormente.

Una vez estén generados los banners por los usuarios existentes lo que hacemos es cargar a HBASE las tablas con los banners y los clientes ejecutando las aplicaciones CargaBannerHBase y CargaClientesHBase para poder acceder con rapidez a los datos desde SPARK.

Mediante Spark Streaming vamos a consultar los eventos que llegan a través del topic de eventos. Para ello ejecutamos la aplicación RecibidorEventos.scala que contiene tanto un spark context, un spark streaming context, un HBASE context y un productor Kafka para devolver el banner correspondiente al usuario del evento.

Esta aplicación se encarga de importar las tablas de HBASE de banners y clientes, comprobar si existe el usuario en el sistema y devolver el banner para el usuario; en caso de que no exista el usuario lo que hacemos es con la tabla de banners de HBASE calcular el banner que le correspondería al usuario del evento y devolverlo por Kafka. En el caso de que no se ajustara a ningún perfil de usuario se devolvería un banner genérico.

Además, el banner generado será devuelto cada X reproducciones de cada usuario, por lo que cada vez que llegue un evento de un usuario se irá escribiendo en una columna de la tabla de HBASE correspondiente al usuario el número de reproducciones que lleva, en el caso de que llegue al número establecido se procederá a devolver el banner correspondiente.

Una vez los resultados se van metiendo en una cola de KAFKA, habría que implementar, por ejemplo, otro agente FLUME para devolver los eventos generados por la aplicación al servidor de JMS.

## Generación de recomendaciones.

En el caso de la generación de recomendaciones, lo primero que tenemos que hacer es generar la tabla en HBASE artistWithName que tendrá el artista y su correnpondiente nombre. Esta tabla se creará a partir de un directorio de HDFS generado en batch anteriormente.

Mediante Spark Streaming vamos a consultar los eventos que llegan a través del topic de eventos. Para ello ejecutamos la aplicación HacerRecomendacion.scala que contiene tanto un spark context, un spark streaming context, un HBASE context y un productor Kafka para devolver la recomendación correspondiente al usuario del evento.

Lo primero que hacemos es cargar el modelo calculado mediante su proceso batch.

Para cada evento se genera una recomendación, para ello se obtiene de la tabla artistWithName el id del artista que está escuchando el usuario en ese momento, se transformaría en un id conocido por el sistema y llamaríamos al método transform()que es el que calcula las recomendaciones.

De las recomendaciones generadas nos quedaríamos con la que mayor rating tiene y es la que devolveríamos a KAFKA que como anteriormente podría dejar a otro agente FLUME para que se lo devuelva al sistema.