

14 DE ENERO DE 2018

PRÁCTICA

BIG DATA ARCHITECTURE

JORGE DEBRÁN PÉREZ

DISEÑO

ANALÍTICA

Análisis de clientes a partir de datos obtenidos en tiempo real y batch, utilizando datos de consumo provenientes de facturación, datos de interacción provenientes del call center, de sus redes sociales y de la web, y datos socioeconómicos obtenidos de la web a partir de datos demográficos del cliente.

A partir de este análisis, realizaremos un tablón para facilitar su visualización:

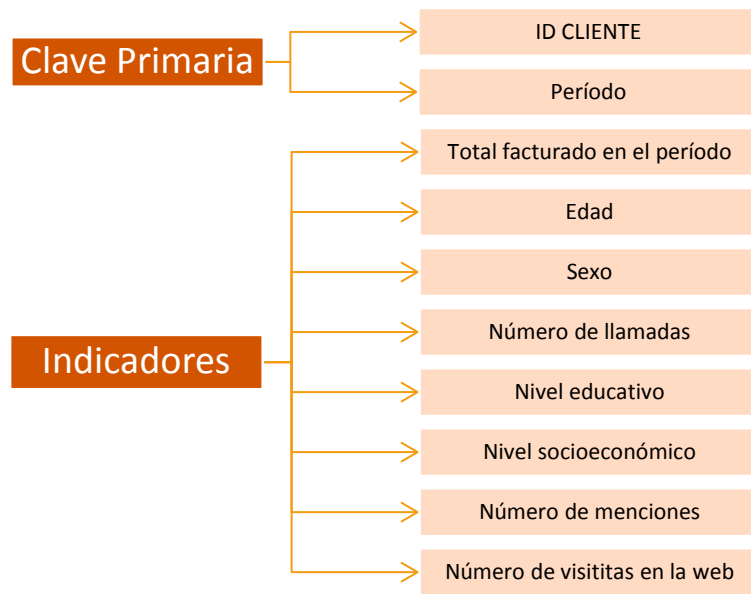


Tabla 1. Tablón CLIENTES con sus indicadores

MODELO LÓGICO Y MODELO FÍSICO SOBRE HIVE

Para poder realizar este tablón de clientes, lo primero que haremos será definir el modelo lógico:

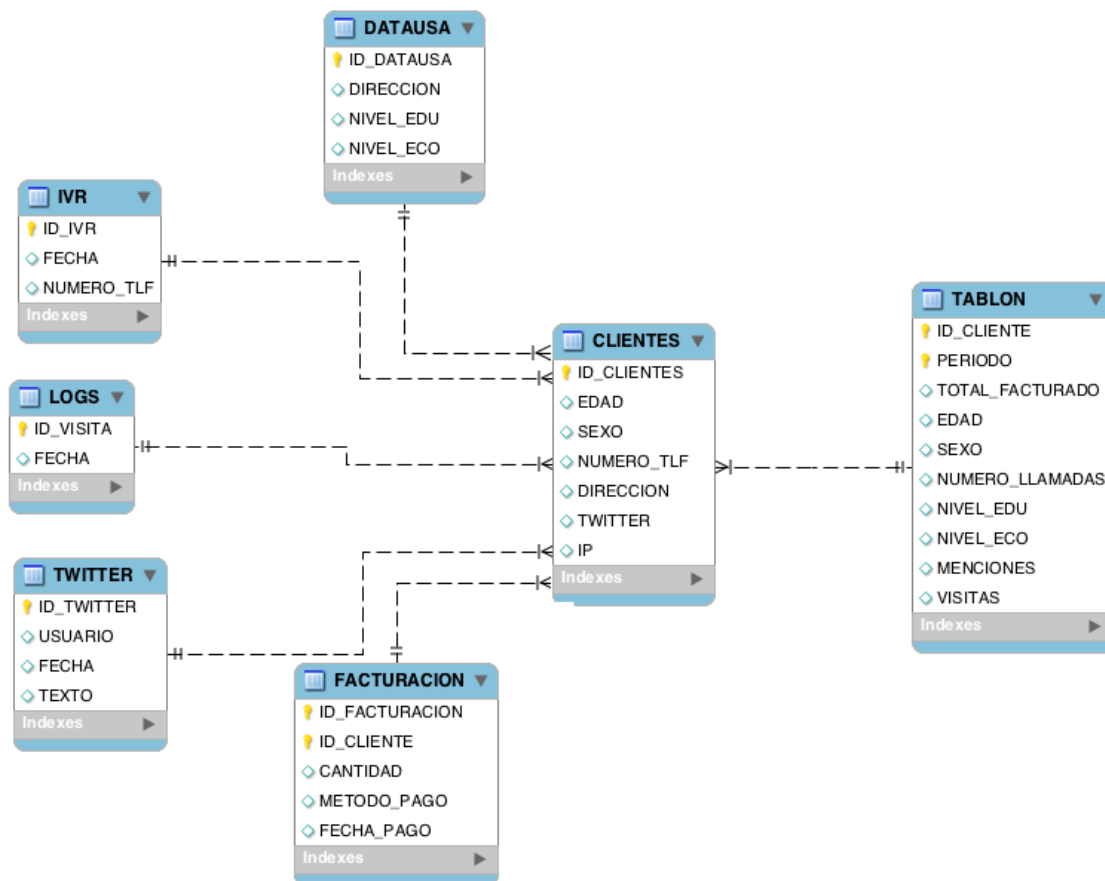


Ilustración 1. Modelo lógico

Para aplicar el modelo lógico utilizaremos el siguiente modelo físico en Hive:

- **CLIENTES**: será una tabla externa, en formato ORC con compresión, particionada por el campo dirección y con 3 buckets sobre el campo ID_CLIENTE.
- **FACTURACION**: será una tabla externa, en formato ORC con compresión, particionada por el campo fecha (año, mes y día) y con 3 buckets sobre el campo ID_CLIENTE.
- **IVR**: será una tabla externa, en formato ORC con compresión, particionada por el campo fecha (año, mes y día) y con 3 buckets sobre el campo NUMERO_TLF.
- **LOGS**: será una tabla externa, en formato ORC con compresión, particionada por el campo fecha (año, mes y día) y con 3 buckets sobre el campo IP.
- **TWITTER**: será una tabla externa, en formato ORC con compresión, particionada por el campo fecha (año, mes y día) y con 3 buckets sobre el campo USUARIO.
- **DATAUSA**: será una tabla externa, en formato ORC con compresión y particionada por el campo dirección.
- **TABLON**: será una tabla interna, en formato ORC con compresión, particionada por el campo fecha (año, mes y día) y con 3 buckets sobre el campo ID_CLIENTE.

Se debería realizar un plan de pruebas para comprobar que el particionado elegido y, sobre todo, el número de buckets es correcto.

FUENTES DE DATOS ESTRUCTURADOS Y “NO ESTRUCTURADOS”

A continuación, debemos identificar de donde podemos traer cada dato y con qué herramienta:

Destino	Origen	Herramienta	Transformación
ID_CLIENTE	Base de datos: ODS Tabla: ODS_HC_CLIENTES	Sqoop	
PERIODO	Base de datos: ODS Tablas: ODS_HC_FACTURAS (FC_PAGO), ODS_HC_LLAMADAS (FC_INICIO)	Sqoop	
TOTAL_FACTURADO	Base de datos: ODS Tabla: ODS_HC_FACTURAS	Sqoop	Será un valor calculado utilizando el campo CANTIDAD
EDAD	Base de datos: ODS Tabla: ODS_HC_CLIENTES	Sqoop	Será un valor calculado a partir de la fecha de nacimiento (campo FC_NACIMIENTO)
SEXO	Base de datos: ODS Tabla: ODS_HC_CLIENTES	Sqoop	
NUMERO_LLAMADAS	Base de datos: ODS Tabla: ODS_HC_LLAMADAS	Sqoop	Será un valor calculado del total de valores
NIVEL_EDU	Data USA (datausa.io)	API	
NIVEL_ECO	Data USA (datausa.io)	API	
MENCIONES	Twitter API	Flume	Será un valor calculado del total de valores
VISITAS	Logs Apache	API	Será un valor calculado del total de valores

Tabla 2. Fuente de datos

Utilizaremos la herramienta Sqoop contra la base de datos ODS en un primer momento para traernos toda la base de datos. Después, establecemos la carga periódica de forma diaria y utilizaremos Sqoop para traernos los datos de forma incremental.

Por otro lado, crearemos una API para traernos los datos de Data USA en formato csv. Estos datos los cargaremos cuando sean publicados, además podremos hacer una comprobación de cuando se han añadido nuevos datos, esta comprobación se podrá hacer de forma mensual. También generaremos una API para parsear los logs de Apache y realizaremos la carga de los datos de forma diaria.

Por último, utilizaremos Flume para traernos en tiempo real a través de la API de Twitter.

Todos los datos de las diferentes fuentes aterrizarán en Raw Data y estableceremos una política de vida útil del dato en su etapa de Staging gracias a Hive. Posteriormente los datos de Staging los transformaremos en el modelo lógico descrito anteriormente utilizando las tablas externas. Para finalizar, se realizará el cálculo de las diferentes KPIs que necesiten ser calculadas y llevando a cabo el tablón con las mismas y guardado como tabla gestiona en Hive.

CONSTRUCCIÓN

CARGA DE DATOS ESTRUCTURADOS

Para realizar la carga de datos hemos utilizado el fichero sqoop.txt:

```

scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_DM_AGENTES_CC --target-dir /kcbda/RAW/ODS_DM_AGENTES_CC -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_DM_AGENTES_PRV --target-dir /kcbda/RAW/ODS_DM_AGENTES_PRV -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_DM_CANALES -
--target-dir /kcbda/RAW/ODS_DM_CANALES -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_DM_CICLOS_FACTURACION --target-dir /kcbda/RAW/ODS_DM_CICLOS_FACTURACION -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_DM_CIUDADES_ESTADOS --target-dir /kcbda/RAW/ODS_DM_CIUDADES_ESTADOS -m 1 -
z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_DM_COMPANYAS
--target-dir /kcbda/RAW/ODS_DM_COMPANYAS -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_DM_DEPARTAMENTOS_CC --target-dir /kcbda/RAW/ODS_DM_DEPARTAMENTOS_CC -m 1 -
z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_DM_FASES --
target-dir /kcbda/RAW/ODS_DM_FASES -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_DM_METODOS_PAGO --target-dir /kcbda/RAW/ODS_DM_METODOS_PAGO -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_DM_PAISES --
target-dir /kcbda/RAW/ODS_DM_PAISES -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_DM_PRODUCTOS
--target-dir /kcbda/RAW/ODS_DM_PRODUCTOS -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_DM_PROFESIONES --target-dir /kcbda/RAW/ODS_DM_PROFESIONES -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_DM_SEXOS --
target-dir /kcbda/RAW/ODS_DM_SEXOS -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_HC_CLIENTES
--target-dir /kcbda/RAW/ODS_HC_CLIENTES -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_HC_DIRECCIONES --target-dir /kcbda/RAW/ODS_HC_DIRECCIONES -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_HC_FACTURAS
--target-dir /kcbda/RAW/ODS_HC_FACTURAS -m 3 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_HC_LLAMADAS
--target-dir /kcbda/RAW/ODS_HC_LLAMADAS -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table
ODS_HC_PROVISIONES --target-dir /kcbda/RAW/ODS_HC_PROVISIONES -m 1 -z
scoop import --connect jdbc:mysql://172.17.0.3:3306/ODS --driver
com.mysql.jdbc.Driver --username root --password root --table ODS_HC_SERVICIOS
--target-dir /kcbda/RAW/ODS_HC_SERVICIOS -m 1 -z

```

Fichero 1. scoop.txt

Destacar que para realizar la carga de la tabla ODS_HC_FACTURAS hemos modificado el valor del mapper estableciéndole a 3. Para todas las cargas se ha habilitado la compresión por defecto gzip, hubiera sido interesante habilitar la compresión con Snappy:

```
--compression-codec snappy
```



Todos los datos se han traído en texto plano aunque sería interesante utilizar el formato Avro:

```
--as-avrodatafile
```

Por último, para poder realizar las cargas incrementales deberemos utilizar los siguientes parámetros:

```
--incremental lastmodified  
--check-column COLUMN_CHECK
```

Una vez cargados los datos con sqoop, crearemos en Hive el área de Raw Data con el fichero kcbda_raw.sql:

```
create database kcbda_raw;  
create database kcbda_staging;  
create database kcbda_trusted;  
  
use kcbda_raw;  
  
CREATE EXTERNAL TABLE ODS_DM_AGENTES_CC (  
    ID_AGENTE_CC int,  
    DE_AGENTE_CC string,  
    FC_INSERT timestamp,  
    FC_MODIFICATION timestamp  
)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","  
STORED AS TEXTFILE  
LOCATION "/kcbda/RAW/ODS_DM_AGENTES_CC";  
  
CREATE EXTERNAL TABLE ODS_DM_AGENTES_PRV (  
    ID_AGENTE_PRV int,  
    DE_AGENTE_PRV string,  
    FC_INSERT timestamp,  
    FC_MODIFICATION timestamp  
)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","  
STORED AS TEXTFILE  
LOCATION "/kcbda/RAW/ODS_DM_AGENTES_PRV";  
  
CREATE EXTERNAL TABLE ODS_DM_CANALES (  
    ID_CANAL int,  
    DE_CANAL string,  
    FC_INSERT timestamp,  
    FC_MODIFICATION timestamp  
)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","  
STORED AS TEXTFILE  
LOCATION "/kcbda/RAW/ODS_DM_CANALES";  
  
CREATE EXTERNAL TABLE ODS_DM_CICLOS_FACTURACION (  
    ID_CICLO_FACTURACION int,  
    DE_CICLO_FACTURACION string,  
    FC_INSERT timestamp,  
    FC_MODIFICATION timestamp  
)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","  
STORED AS TEXTFILE  
LOCATION "/kcbda/RAW/ODS_DM_CICLOS_FACTURACION";  
  
CREATE EXTERNAL TABLE ODS_DM_CIUDADES_ESTADOS (  
    ID_CIUADAD_ESTADO int,  
    DE_CIUADAD string,  
    DE_ESTADO string,  
    ID_PAIS int,  
    FC_INSERT timestamp,  
    FC_MODIFICATION timestamp  
)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","  
STORED AS TEXTFILE
```

```

LOCATION "/kcbda/RAW/ODS_DM_CIUDADES_ESTADOS";

CREATE EXTERNAL TABLE ODS_DM_COMPANYAS (
  ID_COMPANYA int,
  DE_COMPANYA string,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_DM_COMPANYAS";

CREATE EXTERNAL TABLE ODS_DM_DEPARTAMENTOS_CC (
  ID_DEPARTAMENTO_CC int,
  DE_DEPARTAMENTO_CC string,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_DM_DEPARTAMENTOS_CC";

CREATE EXTERNAL TABLE ODS_DM_FASES (
  ID_FASE int,
  DE_FASE string,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_DM_FASES";

CREATE EXTERNAL TABLE ODS_DM_METODOS_PAGO (
  ID_METODO_PAGO int,
  DE_METODO_PAGO string,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_DM_METODOS_PAGO";

CREATE EXTERNAL TABLE ODS_DM_PAISES (
  ID_PAIS int,
  DE_PAIS string,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_DM_PAISES";

CREATE EXTERNAL TABLE ODS_DM_PRODUCTOS (
  ID_PRODUCTO int,
  DE_PRODUCTO string,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_DM_PRODUCTOS";

CREATE EXTERNAL TABLE ODS_DM_PROFESIONES (
  ID_PROFESION int,
  DE_PROFESION string,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)

```



```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_DM_PROFESIONES";
```

```
CREATE EXTERNAL TABLE ODS_DM_SEXOS (
  ID_SEXO int,
  DE_SEXO string,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_DM_SEXOS";
```

```
CREATE EXTERNAL TABLE ODS_HC_CLIENTES (
  ID_CLIENTE int,
  NOMBRE_CLIENTE string,
  APELLIDOS_CLIENTE string,
  NUMDOC_CLIENTE string,
  ID_SEXO int,
  ID_DIRECCION_CLIENTE int,
  TELEFONO_CLIENTE bigint,
  EMAIL string,
  FC_NACIMIENTO date,
  ID_PROFESION int,
  ID_COMPANYA int,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_HC_CLIENTES";
```

```
CREATE EXTERNAL TABLE ODS_HC_DIRECCIONES (
  ID_DIRECCION int,
  DE_DIRECCION string,
  DE_CP int,
  ID_CIUADAD_ESTADO int,
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_HC_DIRECCIONES";
```

```
CREATE EXTERNAL TABLE ODS_HC_FACTURAS (
  ID_FACTURA int,
  ID_CLIENTE int,
  FC_INICIO timestamp,
  FC_FIN timestamp,
  FC_ESTADO timestamp,
  FC_PAGO timestamp,
  ID_CICLO_FACTURACION int,
  ID_METODO_PAGO int,
  CANTIDAD decimal(4,2),
  FC_INSERT timestamp,
  FC_MODIFICATION timestamp
)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_HC_FACTURAS";
```

```
CREATE EXTERNAL TABLE ODS_HC_LLAMADAS (
  ID_LLAMADA int,
  ID_IVR int,
  TELEFONO_LLAMADA bigint,
  ID_CLIENTE int,
  FC_INICIO_LLAMADA timestamp,
```




```

    FC_FIN_LLAMADA timestamp,
    ID_DEPARTAMENTO_CC int,
    FLG_TRANSFERIDO tinyint,
    ID_AGENTE_CC int,
    FC_INSERT timestamp,
    FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_HC_LLAMADAS";

CREATE EXTERNAL TABLE ODS_HC_PROVISIONES (
    ID_PROVISION int,
    ID_ORDER int,
    ID_SERVICIO int,
    ID_FASE int,
    ID_AGENTE_PRV int,
    FC_INICIO timestamp,
    FC_FIN timestamp,
    FC_INSERT timestamp,
    FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_HC_PROVISIONES";

CREATE EXTERNAL TABLE ODS_HC_SERVICIOS (
    ID_SERVICIO int,
    ID_CLIENTE int,
    ID_PRODUCTO int,
    PUNTO_ACCESO string,
    ID_CANAL int,
    ID_AGENTE int,
    ID_DIRECCION_SERVICIO int,
    FC_INICIO timestamp,
    FC_INSTALACION timestamp,
    FC_FIN timestamp,
    FC_INSERT timestamp,
    FC_MODIFICATION timestamp
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE
LOCATION "/kcbda/RAW/ODS_HC_SERVICIOS";

```

Fichero 2. kcbda_raw.sql

Para pasar estos datos a Staging, podemos crear por cada tabla en Raw Data, unas tablas externas nuevas en la localización de Staging de HDFS con las mismas columnas que tenemos en Raw Data y añadir una nueva columna donde ponemos el ciclo de vida o el día de carga:

```

use kcbda_staging;

CREATE EXTERNAL TABLE ODS_DM_AGENTES_CC
LIKE kcbda_raw.ODS_DM_AGENTES_CC
ROW FORMAT DELIMITED FIELDS TERMINATED BY "|"
STORED AS ORC
LOCATION "/kcbda/STAGING/ODS_DM_AGENTES_CC";

ALTER TABLE ODS_DM_AGENTES_CC ADD COLUMNS (dt string);

TRUNCATE TABLE ODS_DM_AGENTES_CC;

INSERT INTO TABLE ODS_DM_AGENTES_CC SELECT *, "2018-01-12" AS dt FROM
kcbda_raw.ODS_DM_AGENTES_CC;

```

CARGA DE DATOS “NO ESTRUCTURADOS”

Para cargar datos de Data USA, hemos utilizado web scrapping y construido una pequeña API para poder guardar los csv que se obtienen de su propia API en el área de Raw Data en HDFS. Este proceso le podemos ver en el fichero Data USA API.ipynb:

```
import urllib3
urllib3.disable_warnings()
http = urllib3.PoolManager()
url = "https://datausa.io/about/attributes/geo/state/1/"
response = http.request('GET', url)

from bs4 import BeautifulSoup
soup = BeautifulSoup(response.data, 'html.parser')
table=soup.find('table', class_='geo')

IDs=[]
Names=[]
for row in table.findAll("tr"):
    cells = row.findAll("td")
    if len(cells)==2:
        IDs.append(cells[0].find(text=True))
        Names.append(cells[1].find(text=True))

from pyhive import hive
conn = hive.Connection(host='sandbox.hortonworks.com', port=10000)
cursor = conn.cursor()

cursor.execute('use kcbda_raw')
cursor.execute('CREATE EXTERNAL TABLE DATAUSA_STATE (ID_STATE string, DE_STATE
string) ROW FORMAT DELIMITED FIELDS TERMINATED BY "," STORED AS TEXTFILE
LOCATION "/kcbda/RAW/DATAUSA_STATE"')

query = "INSERT INTO TABLE DATAUSA_STATE VALUES "
for id, name in list(zip(IDs, Names)):
    query += "(" + str(id) + ", " + str(name) + "),"
query = query[:-1]
cursor.execute(query)

import hdfs
from hdfs import InsecureClient
client = InsecureClient('http://sandbox.hortonworks.com:50070', user='hdfs',
timeout=1000)

url =
'http://api.datausa.io/api/csv/?show=geo&sumlevel=state&required=avg_wage&year
=latest'
to_path = '/kcbda/RAW/DATAUSA_AVG/2015.csv'
response = http.request('GET', url)
client.write(to_path, response.data)

url =
'http://api.datausa.io/api/csv/?show=geo&sumlevel=state&required=high_school_g
raduation&year=latest'
to_path = '/kcbda/RAW/DATAUSA_HIGH_SCHOOL/2017.csv'
response = http.request('GET', url)
client.write(to_path, response.data)
```

Fichero 3. Data USA API.ipynb

En este proceso deberíamos incluir los procesos de carga de logs de Apache creando una API tal como se ha visto en clase. Además, deberemos de realizar el proceso de carga de la API de Twitter a través de la herramienta Flume.

TABLÓN CLIENTES

Lo primero que debemos hacer para poder construir el tablón deberemos pasar los datos de Staging a la Trusted Zone, aunque en nuestro caso les vamos a pasar desde Raw Data.

Para los datos estructurados este paso le tenemos en el fichero kcbda_trusted.sql:

```
use kcbda_trusted;

CREATE EXTERNAL TABLE CLIENTES (
  ID_CLIENTE int,
  FC_NACIMIENTO date,
  DE_SEXO string,
  TELEFONO_CLIENTE bigint,
  TWITTER string,
  IP string
)
PARTITIONED by (DE_ESTADO string)
CLUSTERED BY (ID_CLIENTE) INTO 3 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY "|"
STORED AS ORC
LOCATION "/kcbda/TRUSTED/CLIENTES";

INSERT INTO TABLE CLIENTES PARTITION (DE_ESTADO)
SELECT
  C.ID_CLIENTE,
  C.FC_NACIMIENTO,
  S.DE_SEXO,
  C.TELEFONO_CLIENTE,
  "@USER",
  "XXX.XXX.XXX.XXX",
  CE.DE_ESTADO
FROM kcbda_raw.ODS_HC_CLIENTES AS C
INNER JOIN kcbda_raw.ODS_DM_SEXOS S ON C.ID_SEXO = S.ID_SEXO
INNER JOIN kcbda_raw.ODS_HC_DIRECCIONES D ON C.ID_DIRECCION_CLIENTE =
D.ID_DIRECCION
INNER JOIN kcbda_raw.ODS_DM_CIUDADES_ESTADOS CE ON D.ID_CIUADAD_ESTADO =
CE.ID_CIUADAD_ESTADO;

CREATE EXTERNAL TABLE FACTURAS (
  ID_FACTURA int,
  ID_CLIENTE int,
  CANTIDAD decimal(4,2),
  DE_METODO_PAGO string
)
PARTITIONED by (FC_PAGO_Y string, FC_PAGO_M string, FC_PAGO_D string)
CLUSTERED BY (ID_CLIENTE) INTO 3 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY "|"
STORED AS ORC
LOCATION "/kcbda/TRUSTED/FACTURAS";

INSERT INTO TABLE FACTURAS PARTITION (FC_PAGO_Y, FC_PAGO_M, FC_PAGO_D)
SELECT
  F.ID_FACTURA,
  F.ID_CLIENTE,
  F.CANTIDAD,
  MP.DE_METODO_PAGO,
  YEAR(F.FC_PAGO) AS FC_PAGO_Y,
  MONTH(F.FC_PAGO) AS FC_PAGO_M,
  DAY(F.FC_PAGO) AS FC_PAGO_D
FROM kcbda_raw.ODS_HC_FACTURAS AS F
INNER JOIN kcbda_raw.ODS_DM_METODOS_PAGO MP ON F.ID_METODO_PAGO =
MP.ID_METODO_PAGO;
```

Fichero 4. kcbda_trusted.sql

A continuación, cargaremos los datos no estructurados, en nuestro caso solo podremos cargar los datos de Data USA. Esta carga la tenemos en el fichero data usa_trusted.sql:

```
CREATE EXTERNAL TABLE DATAUSA_HIGH_SCHOOL (
  year int,
  geo_name string,
  geo string,
  high_school_graduation decimal(4,3)
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
LOCATION "/kcbda/RAW/DATAUSA_HIGH_SCHOOL"
tblproperties("skip.header.line.count"="1");

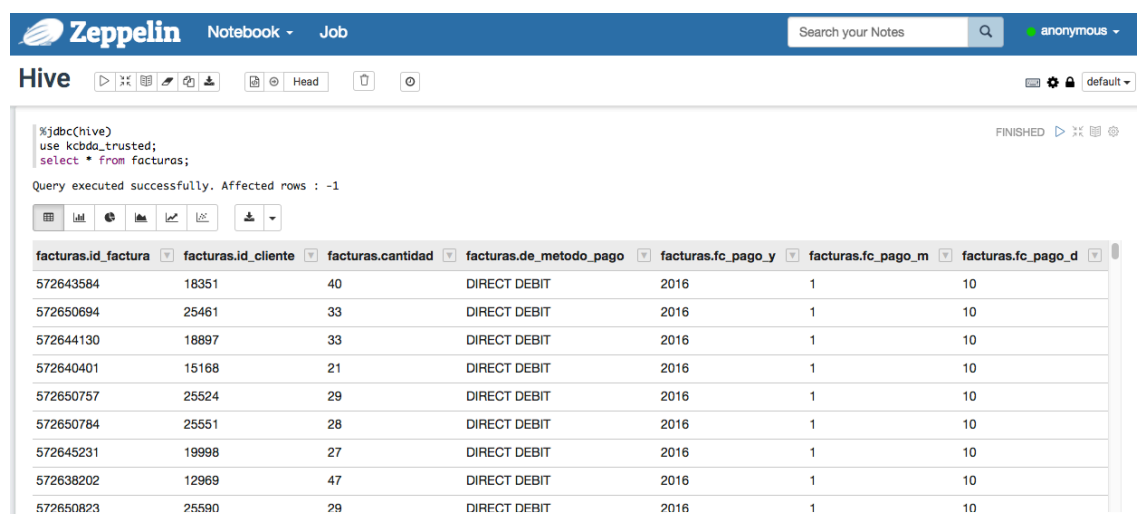
CREATE EXTERNAL TABLE DATAUSA_AVG (
  year int,
  geo_name string,
  geo string,
  avg_wage decimal(6,1)
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
LOCATION "/kcbda/RAW/DATAUSA_AVG"
tblproperties("skip.header.line.count"="1");

CREATE EXTERNAL TABLE DATAUSA (
  high_school_graduation decimal(4,3),
  avg_wage decimal(6,1)
)
PARTITIONED BY (DE_ESTADO string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY "|"
STORED AS ORC
LOCATION "/kcbda/TRUSTED/DATAUSA";

INSERT INTO TABLE DATAUSA PARTITION (DE_ESTADO)
SELECT
DHS.high_school_graduation,
DAVG.avg_wage,
UPPER(DS.DE_STATE) AS DE_ESTADO
FROM kcbda_raw.DATAUSA_STATE AS DS
INNER JOIN kcbda_raw.DATAUSA_HIGH_SCHOOL DHS ON DS.ID_STATE = DHS.geo
INNER JOIN kcbda_raw.DATAUSA_AVG DAVG ON DS.ID_STATE = DAVG.geo;
```

Fichero 5.data usa_trusted.sql

La construcción de la Trusted Zone la podemos ver desde Zeppelin:



Zeppelin Notebook - Job

Search your Notes

anonymous

Hive

Query executed successfully. Affected rows : -1

facturas.id_factura	facturas.id_cliente	facturas.cantidad	facturas.de_metodo_pago	facturas.fc_pago_y	facturas.fc_pago_m	facturas.fc_pago_d
572643584	18351	40	DIRECT DEBIT	2016	1	10
572650694	25461	33	DIRECT DEBIT	2016	1	10
572644130	18897	33	DIRECT DEBIT	2016	1	10
572640401	15168	21	DIRECT DEBIT	2016	1	10
572650757	25524	29	DIRECT DEBIT	2016	1	10
572650784	25551	28	DIRECT DEBIT	2016	1	10
572645231	19998	27	DIRECT DEBIT	2016	1	10
572638202	12969	47	DIRECT DEBIT	2016	1	10
572650823	25590	29	DIRECT DEBIT	2016	1	10

Ilustración 2. Facturas en la Trusted Zone

A continuación, deberemos de ir generando las diferentes KPIs. Por ejemplo, en facturación se realizaría de la siguiente manera:

```
SELECT
F.ID_CLIENTE, F.FC_PAGO_Y, F.FC_PAGO_M, F.FC_PAGO_D, SUM(F.CANTIDAD)
FROM FACTURAS AS F
GROUP BY F.FC_PAGO_Y, F.FC_PAGO_M, F.FC_PAGO_D, F.ID_CLIENTE
```

Así iríamos completando todas las KPIs y poder generar el tablón, pero debido a continuos errores de memoria en Hive (java.lang.OutOfMemoryError) no he podido completar.

El tablón le generaríamos con la siguiente consulta:

```
CREATE TABLE TABLON_CLIENTES (
  ID_CLIENTE int,
  TOTAL_FACTURADO decimal,
  EDAD int,
  SEXO string,
  NUMERO_LLAMADAS int,
  NIVEL_EDU decimal,
  NIVEL_ECO decimal,
  MENCIONES int,
  VISITAS int
)
PARTITIONED by (FC_PAGO_Y string, FC_PAGO_M string, FC_PAGO_D string)
CLUSTERED BY (ID_CLIENTE) INTO 3 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY "|"
STORED AS ORC
LOCATION "/kcbda/TRUSTED/TABLON_CLIENTES";
```

Este tablón sería muy interesante llevarlo a alguna herramienta OLAP y desarrollar cubos:

