## Diseño:

1. En nuestro caso queremos usar Big Data para una compañía de telecomunicaciones, por lo que lo primero sería definir los objetivos que se quieren conseguir y para los que el Big Data puede ser útil (centrándonos en analíticas de clientes):

* Rentabilidad por cliente: Puede ser interesante medir la facturación de los clientes para determinar qué clientes son más rentables (clientes VIP). Denominaré a este indicador **TotalFact**, y será un número decimal que indica el total facturado por cliente en el periodo. Aquí también podríamos tener un indicador sobre la segmentación socioeconómica a la que pertenece el cliente, basada en la profesión. Para la práctica voy a suponer que tengo un fichero proveniente de datos abiertos de alguna organización (tipo instituto nacional de estadística) donde se indica profesión - sueldo medio para dicha profesión.
* Satisfacción del cliente: Podríamos intentar medir el grado de satisfacción de un cliente. Para saber esto podemos fijarnos en el indicador **LLaCCClaims** (llamadas al Call Center al departamento CLAIMS, es decir, a quejas). Este indicador lo dividiré en **LLCCClaimsTrue**, que son las llamadas que terminaron resolviendo el problema del cliente de forma positiva y **LLCCClaimsFalse**, que son las llamadas en las que el cliente no quedó satisfecho. Podemos también crear un indicador basado en la actividad del cliente en las redes sociales. Como todo este tema se escapa del alcance de la práctica, para simplificar y poder aplicar algo de lo que hemos visto, voy a suponer que tengo un fichero de log, que parseo y del que obtengo dos campos relevantes, el email del cliente y valoración (de 0 a 5 estrellas). En la vida real se podría usar Kafka para hacer un seguimiento en tiempo real de la actividad del cliente en las redes sociales. El indicador que obtenga de esta fuente voy a llamarlo **ValoracionCli**.
* Fidelización del cliente: Voy a establecer un indicador los productos contratados por un cliente, que llamaré **TotalPoductos**.

1. Definición del modelo lógico y físico: Para obtener los indicadores definidos en el punto anterior, necesito datos del cliente: número de teléfono y profesión para obtener los datos socioeconómicos. Me voy a traer también el Email, que he decidido que será la información con la que crucemos los datos de la valoración del cliente. También necesito datos de las llamadas al Call Center, de las facturas y de los servicios. Tendría el siguiente modelo lógico:

Tablón

Cliente

|  |
| --- |
| Id. Cliente |
| Profesion |
| Email |

|  |
| --- |
| Id. cliente |
| Periodo |
| Total facturado |
| Total LlaCCClaimsTrue |
| Total LlaCCClaimsFalse |
| Total Productos |
| Sueldo medio |
| Valoración media |

Llamadas (al dpto. CLAIMS)

|  |
| --- |
| Teléfono |
| F. Inicio |
| F. Fin |
| Acción |
| Resultado |

Facturas

|  |
| --- |
| Id. Cliente |
| F. Inicio |
| F. Fin |
| Cantidad |

Servicios

|  |
| --- |
| Id. Cliente |
| F. Contratación |
| Producto |

Actividad Web

|  |
| --- |
| Email |
| Valoracion |

Nivel Socioeconomico

|  |
| --- |
| Profesión |
| Sueldo medio |

El tablón se cargará con periodicidades mensuales, como la valoración del usuario en la web la voy a cargar diariamente (igual esto no tiene demasiado sentido, pero es para orientar un poco la práctica a posibles situaciones que nos podamos encontrar en otros escenarios en los que pudiera tener más sentido hacerlo así), el valor reflejado en el indicador del tablón es una media de todos los valores que aparezcan en cada día.

**Modelo físico en Hive**: Voy a prescindir de la tabla de llamadas del call center básicamente porque no cruzan los datos de los teléfonos en la tabla llamadas con los teléfonos en la tabla clientes, aunque en un entorno real esto no pasaría y nos traeríamos también esta tabla. Otra consideración a tener en cuenta es la posibilidad de que el tablón se encuentre en otra parte, estilo HBase con mejores características para el acceso por parte de los usuarios a las KPI’s. En nuestra práctica estará todo en Hive. Lo que voy a definir a continuación es cómo quedaría nuestra parte de la TRUSTED ZONE.

Clientes Facturas Servicios Act. Web Nvl. economico

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Id\_Cliente |  | Id\_cliente |  | Id\_cliente |  | Email |  | Profesion |
| Email |  | F. Inicio |  | F. contratacion |  | F. Valoracion |  | F. insert |
| Profesion |  | F. Fin |  | Producto |  | Valoración |  | Sueldo medio |
| Fc\_clientes |  | Cantidad |  |  |  | Fc\_actividad |  | Fc\_profesiones |

Tablón

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id. Cliente | Periodo | TotalFact | TotalProductos | ValoracionCli | SueldoMedio |

Realizaremos un particionamiento de las tablas para mejorar la eficiencia de los JOINs como sigue:

* La tabla Clientes, particionamos por el id\_cliente mod 10.
* La tabla Facturas, por el mes de la fecha de la factura (No incluyo el año para tener un número máximo de particiones, en este caso tendría 12)
* La tabla Servicios, por el mes de la fecha de contratación, haciendo un razonamiento similar al de la tabla anterior.
* El tablón lo particionaré también por el id\_cliente mod 10.

El formato que elijo en este caso es TEXTFILE, porque me pareció el más sencillo para empezar a probar. En realidad, no disponemos de suficientes datos como para realizar las pruebas necesarias. En un entorno real se prepararía un piloto con un set de pruebas suficiente para medir los tiempos usando un formato u otro y con las conclusiones se tomaría la decisión.

1. Partimos de un servidor mySQL que tiene los datos del operacional, ahí es donde están mis tablas del DataWarehouse que servirán como una de las fuentes de los datos. Lo primero que habría que hacer es mediante sqoop cargar los datos de algunas de las tablas de ODS en el área Raw Data. Con los indicadores elegidos cargaré la tabla ODS\_HC\_CLIENTES, aunque sólo los datos referidos a ID\_CLIENTE, EMAIL y PROFESIÓN. También necesitaría los campos TELEFONO\_LLAMADA, FC\_INICIO\_LLAMADA, FC\_FIN\_LLAMADA de la tabla ODS\_HC\_LLAMADAS, filtrados por el ID\_DEPARTAMENTO = 4, que corresponde con el departamento de quejas. Con los datos de los que dispongo no puedo determinar la acción ni el resultado. El resultado se podría anotar después de la llamada, con una breve encuesta en la que el cliente indique la satisfacción con el resultado de la resolución de su problema. De todas formas, podríamos sacar más indicadores de la tabla de llamadas al Call Center, como las ventas que se llevan a cabo por ese canal, llamadas a atención al cliente que nos podrían dar información sobre la efectividad del servicio de atención al cliente, llamadas a soporte, que también podrían indicar el nivel de incidencias, etc… pero en la práctica y con los datos que disponemos no vamos a poder sacar indicadores de esta tabla, ya que como comenté antes, los teléfonos de la tabla LLAMADAS no cruzan con los teléfonos de la tabla CLIENTES, por lo que no se puede establecer una relación entre la llamada al call center y el cliente que la realiza. Por tanto, no voy a traerme la tabla de LLAMADAS.

Si me traeré la tabla ODS\_HC\_FACTURAS y la tabla ODS\_HC\_SERVICIOS, aunque sólo me traeré los campos que necesito para calcular los indicadores y que he puesto en el esquema de arriba.

Por otro lado, necesito datos socioeconómicos asociados a la profesión del cliente que sacaré de un hipotético fichero que proporciona el instituto nacional de estadística donde se informa de los sueldos medios de las distintas profesiones (me lo voy a inventar) y también datos sobre la actividad del cliente en redes sociales, que he supuesto que los tengo en un fichero de logs.

En este caso, los datos estructurados provendrán de las tablas del DataWarehouse y los datos “no estructurados” son los que sacaré de la web y del fichero que nos proporciona el instituto nacional de estadística. Voy a identificar el origen de cada uno de los datos de las tablas del punto 2, teniendo en cuenta que al final no cargaremos la tabla del Call Center, por lo que no meteré los datos relacionados con esa tabla.

Destino Origen

|  |  |
| --- | --- |
| Id. cliente | ID\_CLIENTE de la tabla ODS\_HC\_CLIENTES |
| Email | EMAIL de la tabla ODS\_HC\_CLIENTES |
| Profesion | PROFESION de la tabla ODS\_DM\_PROFESIONES |
| Id. Cliente | ID\_CLIENTE de la tabla ODS\_HC\_FACTURAS |
| F. Inicio | FC\_INICIO de la tabla ODS\_HC\_FACTURAS |
| F. Fin | FC\_FIN de la tabla ODS\_HC\_FACTURAS |
| Cantidad | CANTIDAD de la tabla ODS\_HC\_FACTURAS |
| Id. Cliente | ID\_CLIENTE de la tabla ODS\_HC\_SERVICIOS |
| F. Contratacion | FC\_INICIO de la tabla ODS\_HC\_SERVICIOS |
| Producto | ID\_PRODUCTO de la tabla ODS\_HC\_SERVICIOS |
| Email | Email que se extrae del log |
| F. valoracion | Fecha de la valoración extraída del log |
| Valoracion | Dato también extraído del log |
| Profesion | La obtengo del fichero que nos proporciona el instituto nacional de estadística. |
| Sueldo medio | También lo obtengo del fichero. |
| Fc. calculo | Fecha en la que se ha hecho el cálculo de los sueldos, que vendrá también en el fichero. |

**Carga inicial:**

**P0 – P1 – P2:**

* Datos estructurados: cargamos los datos de las tablas ODS\_HC\_CLIENTES, ODS\_HC\_FACTURAS y ODS\_HC\_SERVICIOS de mySQL a HDFS, en lo que será nuestra área RAW y STAGING. Generamos los ficheros CLIENTES\_Timestamp, FACTURAS\_Timestamp, SERVICIOS\_Timestamp. En esta práctica voy a unir estas dos áreas ya que para este caso sólo se me ocurre añadir un timestamp con la fecha de carga y esto lo haré a la hora de traerme los datos del operacional (añado now() en la select que me traerá los datos para tener una columna más con la fecha). En un caso real podríamos tener que diferenciar ambas zonas, para lo que se aplicarían los procesos necesarios para las transformaciones de los datos.
* Datos “no estructurados”: pasamos a HDFS el fichero de las profesiones y el log, conectándonos a HDFS con Python y añadiendo una ‘columna’ más en cada fila con la fecha actual.

**P3:**

* Datos estructurados y “no estructurados”: Cargamos las tablas externas de HIVE, que en esta primera carga serán iguales a lo que tenemos en el paso anterior.

**P4:**

* Datos estructurados y “no estructurados”: Introducimos los datos en tablas gestionadas de Hive, que me servirán para hacer la selects que serán la fuente de la carga del tablón.

**P5:**

* Construyo el tablón. Este tablón será una tabla gestionada de HIVE. Como comenté anteriormente, podría ser interesante llevarnos el tablón a otro entorno o incluso aunque sea dentro de HIVE en otro esquema con permisos especiales para los usuarios que finalmente consultan esos indicadores.

Tras la carga inicial, los datos se irán cargando según la siguiente periodicidad:

* Tabla de ODS\_HC\_CLIENTES, ODS\_HC\_FACTURAS y ODS\_HC\_SERVICIOS, se cargarán todos los meses. Para ello seguimos el siguiente flujo:

**P0 – P1 – P2**:

* Cargamos los datos de mySQL en FACTURAS\_Timestamp, SERVICIOS\_Timestamp (no borro los datos anteriores para mantener un histórico de todas las cargas).

La tabla clientes es un poco especial, porque si no se mantiene un histórico de los clientes en el origen, es decir, en las tablas del DataWarehouse, deberíamos hacer una comparación de los datos anteriores con los nuevos datos para detectar los cambios. Esto es bastante costoso, en nuestro caso vamos a borrar los datos que cargamos con anterioridad y sustituirlos por los nuevos.

**P3:**

* Cargamos los datos a tablas externas de Hive, borrando en este caso las tablas anteriores y creándolas de nuevo con los nuevos registros que pasaremos a procesar.

**P4:**

* Pasamos los nuevos datos a las tablas gestionadas de la TRUSTED ZONE.

**P5:**

* Generamos los nuevos registros en el tablón.
* Fichero de actividad\_web:

**P0 – P1 – P2:**

* Cargamos todos los días el fichero de actividad\_web, volcando en RAW todos los datos a diario, y al final del mes, pasamos a la tabla externa de HIVE la información que se procesará una vez al mes calculando así la media de la puntuación diaria. Este proceso será ligeramente distinto del de las otras tablas, porque al tener periodicidad diaria tendremos que cargar todos los archivos de cada uno de los días del mes.

**P4 y P5:** Igual que en el caso anterior una vez al mes.

* Fichero de profesiones: Este fichero se cargará y procesará una vez al año. Los datos estarán disponibles entonces en RAW durante ese año hasta que se realice una nueva carga que los actualice.

Para hacer la parte de gobierno y orquestación podríamos utilizar oozie. Tendríamos que definirnos un flujo de trabajo en el que tienen que quedar reflejados cada uno de los Px definidos con anterioridad. También habría que determinar las acciones a realizar si se produce algún error en cada uno de los casos, así como las comprobaciones necesarias de que los datos se han cargado correctamente, no se han procesado dos veces…etc.

## Construcción:

Voy a tomar un pequeño set de datos para no sobrecargar la sandbox, me limitaré a traerme 500 registros. Todo esto que voy a describir sería la forma “manual” de cargar los datos. En realidad, necesitaríamos definir un flujo en Oozie, para cada uno de estos pasos, programarlos por ejemplo con Python y ejecutarlos con la periodicidad que corresponda en cada caso.

**Carga inicial:**

**P0 – P1 – P2:**

* Datos estructurados:

Genero el archivo **CLIENTES\_20180101** con el siguiente comando:

sqoop import --connect jdbc:mysql://172.17.0.1/ODS --driver com.mysql.jdbc.Driver --username test --password testpwd --query 'SELECT a.ID\_CLIENTE, a.EMAIL, b.DE\_PROFESION, NOW() FROM ODS\_HC\_CLIENTES a JOIN ODS\_DM\_PROFESIONES b ON a.ID\_PROFESION = b.ID\_PROFESION WHERE $CONDITIONS LIMIT 500' -m 1 --target-dir /kcbda/CLIENTES\_20180101

Para el archivo **FACTURAS\_20180101** usaré el siguiente comando:

sqoop import --connect jdbc:mysql://172.17.0.1/ODS --driver com.mysql.jdbc.Driver --username test --password testpwd --query 'SELECT FAC.ID\_CLIENTE, FAC.FC\_INICIO, FAC.FC\_FIN, FAC.CANTIDAD, NOW() FROM ODS\_HC\_FACTURAS FAC WHERE $CONDITIONS LIMIT 500' -m 1 --target-dir /kcbda/FACTURAS\_20180101

Para el archivo **SERVICIOS\_20180101** ejecuto el siguiente commando:

sqoop import --connect jdbc:mysql://172.17.0.1/ODS --driver com.mysql.jdbc.Driver --username test --password testpwd --query ' SELECT ID\_CLIENTE, FC\_INICIO, ID\_PRODUCTO, NOW() FROM ODS\_HC\_SERVICIOS WHERE $CONDITIONS ORDER BY ID\_CLIENTE ASC LIMIT 500;' -m 1 --target-dir /kcbda/SERVICIOS\_20180101

* Datos no estructurados: Para poder simular este escenario, me he inventado dos ficheros actividad\_web.txt y profesiones.txt. El primero es un fichero en el que cada fila consta de tres valores separados por ; El primero es el email, luego viene la puntuación del usuario, y por último, la fecha en la que se realizó esa puntuación. El segundo fichero tiene también filas de valores separados por ; El primero es la profesión, el segundo el sueldo medio y el tercero la fecha en la que se guardó el dato. Mediante Python voy a leer estos ficheros línea a línea y añadirles un campo más con la fecha actual para pasarlos a HDFS (RAW zone) con la transformación necesaria para mi ejemplo (uno las dos zonas RAW y STAGING). Es decir, a la vez que los paso a HDFS añado la columna extra con la fecha de inserción:

# import de módulos necesarios

import hdfs

import os, sys

import time

from hdfs import InsecureClient

# apertura de sesion con WebHDFS

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

# estos datos podrían pasarse como parámetros del programa

source\_file = "ficheros/actividad\_web.txt"

dest\_file = "actividad\_web"

dest\_path = "/kcbda/datos\_web\_20180101/"

dpath = dest\_path + dest\_file

#apertura del fichero

fo = open (source\_file, 'r')

#bucle de lectura

l = ''

for linea in fo:

l = l + linea + ';' + time.strftime("%d/%m/%y %H:%M:%S")

client.write(dpath,data=l)

Con un programa similar hago la carga del otro fichero, profesiones.txt.

Si tuviéramos que realizar otro tipo de transformaciones y tuviéramos las dos zonas diferenciadas (RAW y STAGING) podríamos pasar los ficheros directamente a HDFS mediante estos comandos y luego allí aplicar las transformaciones necesarias para el área de STAGING:

hdfs –dfs put actividad\_web.txt /kcbda/datos\_web\_20180101

hdfs –dfs put profesiones.txt /kcbda/profesiones\_20180101

**P3:**

* Datos estructurados y “no estructurados”: Creo las tablas externas de hive que contendrán los datos que voy a procesar.

create external table CLIENTES

(id\_cliente bigint,

email string,

profesion string,

fc\_insert timestamp

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

STORED AS TEXTFILE

location "/kcbda/CLIENTES\_20180101";

create external table FACTURAS

(id\_cliente bigint,

f\_inicio timestamp,

f\_fin timestamp,

cantidad decimal,

fc\_insert timestamp

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

STORED AS TEXTFILE

location "/kcbda/FACTURAS\_20180101";

create external table SERVICIOS

(id\_cliente bigint,

f\_contratacion timestamp,

producto bigint,

fc\_insert timestamp)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

STORED AS TEXTFILE

location "/kcbda/SERVICIOS\_20180101";

create external table ACTIVIDAD\_WEB

(email string,

valoracion string,

f\_valoracion string,

fc\_insert string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ';'

STORED AS TEXTFILE

location "/kcbda/datos\_web\_20180101";

create external table PROFESIONES

(profesion string,

sueldo bigint,

fc\_calculo string,

fc\_insert string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ';'

STORED AS TEXTFILE

location "/kcbda/profesiones\_20180101";

**P4:**

* Crearía las tablas gestionadas de hive que serían parecidas a las tablas externas sólo que le añado una nueva columna fc\_clientes para indicar el periodo.

create table clientes\_gestionada

(id\_cliente bigint,

email string,

profesion string,

fc\_clientes string

);

Haría lo mismo para el resto de tablas, que luego poblaría con comandos como este:

insert into table clientes\_gestionada

select

id\_cliente,

email,

profesion,

'2018-01-01'

from clientes;

**P5:**

* Creamos las KPI’s con los datos de los que disponemos en las tablas gestionadas

**Sucesivas cargas, tras la carga inicial:**

A partir de aquí, tenemos que programar las cargas de los datos según la periodicidad que hayamos establecido. Por ejemplo, para la tabla de facturas que dije que cargaría una vez al mes se seguirían los siguientes pasos:

**P0 – P1 – P2:** Cargamos los datos del DataWarehouse correspondientes a las facturas insertadas en el mes en cuestión:

sqoop import --connect jdbc:mysql://172.17.0.1/ODS --driver com.mysql.jdbc.Driver --username test --password testpwd --query 'SELECT FAC.ID\_CLIENTE, FAC.FC\_INICIO, FAC.FC\_FIN, FAC.CANTIDAD, NOW() FROM ODS\_HC\_FACTURAS FAC WHERE CAST(FAC.FC\_INICIO as DATE) >='2018-02-01' AND $CONDITIONS LIMIT 500' -m 1 --target-dir /kcbda/FACTURAS\_20180102

Así tenemos nuestros datos en RAW en un nuevo fichero, sin borrar el anterior.

**P3:** Ahora pasamos los datos a las tablas externas de HIVE. En este caso, borro la tabla anterior y la sustituyo por la nueva con los nuevos datos a procesar. Estos serían los comandos:

drop table FACTURAS

create external table FACTURAS

(id\_cliente bigint,

f\_inicio timestamp,

f\_fin timestamp,

cantidad decimal,

fc\_insert timestamp

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

STORED AS TEXTFILE

location "/kcbda/FACTURAS\_20180201";

En este caso apuntamos el ‘location’ al nuevo fichero exportado con sqoop, es decir, FACTURAS\_20180201.

Haríamos lo mismo para el resto de tablas.

**P4:** Como en las tablas externas tenemos los nuevos registros a procesar, en este caso me limito a traer a mis tablas gestionadas los datos que encuentro en las tablas externas. En este caso no se borraría ninguna tabla, simplemente se cargaría con los nuevos datos, es decir, ejecutaríamos un insert into nombre\_tabla\_gestionada select ‘los datos de la tabla externa correspondiente’.

**P5:** Ejecutaría las selects de los indicadores sobre los nuevos datos. Para hacer pequeño ejemplo de cálculo de dos KPI’s he ejecutado la siguiente query con la que obtendría dos de los indicadores:

select

cli.id\_cliente as id\_cliente

,substr(cli.fc\_clientes,0,7) as periodo

,avg(act.valoracion) as valoracion

,avg(sueldo) as sueldo

from clientes\_gestionada cli

inner join actividad\_web\_gestionada act

on act.email = cli.email

and substr(act.fc\_actividad,0,7) = substr(cli.fc\_clientes,0,7)

inner join profesiones\_gestionada prof

on prof.profesion = cli.profesion

and substr(prof.fc\_profesion,0,7) = substr(cli.fc\_clientes,0,7)

where CAST(cli.fc\_clientes as DATE) >='2018-01-01' and

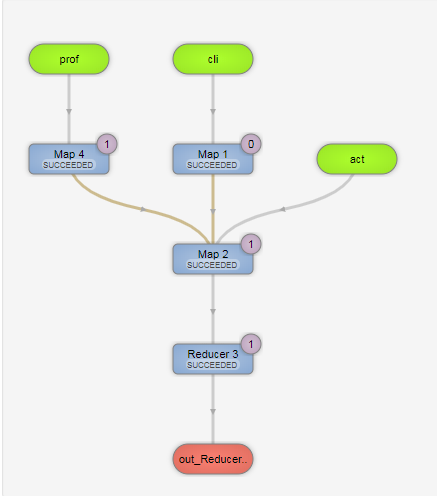
CAST(cli.fc\_clientes as DATE) <='2018-02-01'

group by

cli.id\_cliente,

substr(cli.fc\_clientes,0,7)

He añadido en la cláusula where las fechas que corresponden al periodo que estoy calculando en ese momento. Pero no he conseguido que me devuelvan datos, he debido traerme un conjunto de datos que no cuadra. Lo que sí que he conseguido es ver el siguiente esquema en Tez de cómo se hacen las joins:



Aquí lo interesante sería ahora cambiar los distintos formatos de las tablas, añadir particionamiento y bucketing y lanzar varias veces la query para ver cómo se ejecuta y que combinación es la más óptima.