Este documento se etiqueta como “No compartible” por el hecho de que además de las consignas necesarias para cada parte del laboratorio, algunas de estas partes cuentan con la solución específica para la problemática planteada.

Las soluciones se encuentran en:

- Anexo 1: Lab - Hadoop
- Anexo 2: Lab - Spark
- Anexo 3: Lab - NiFi

Introducción

En el presente documento se les presenta la segunda opción por la que es posible optar para el desarrollo del laboratorio, aquí tendrán a disposición la descripción de la problemática y una descripción breve de los recursos con los cuales cuentan.

Además de las cuestiones a resolver que son específicas para este escenario propuesto, contarán con ejercicios prácticos extras, que serán propios de las unidades de Hadoop y Spark. Estos los pueden encontrar en los anexos correspondientes.

El lab se separa en dos partes:

1. En la sección problemática se detalla una consigna general y una específica para el documento.
2. En los anexos se describirán ejercicios extras que deberán ser completados para las unidades de Hadoop, Spark y NiFi.

Recomendación: realizar primeramente los ejercicios de Hadoop y Spark. ¿Por qué? Porque al atacar a la problemática específica planteada en este documento, tendrán que tener ciertos recursos de los que se hablan en los ejercicios complementarios de Hadoop y Spark. Esto no es obligatorio, pero ciertamente será de ayuda.

Problemática

Consigna general: se debe implementar una solución integral que dé respuesta a las necesidades de información, estas necesidades van desde la ingesta de datos hasta la visualización de los mismos, pasando por etapas de procesamiento y enriquecimiento.

La problemática a resolver se da en un entorno de movimientos de dinero asociados a ventas realizadas, en este escenario contarán con un tópico Kafka ya creado que se encuentra recibiendo mensajes de un productor, lo que deberán realizar es: la ingesta de los mensajes correspondientes a ventas realizadas en tiempo real, el posterior enriquecimiento de los mensajes y luego el cruce de información con registros que tendrán disponibles.

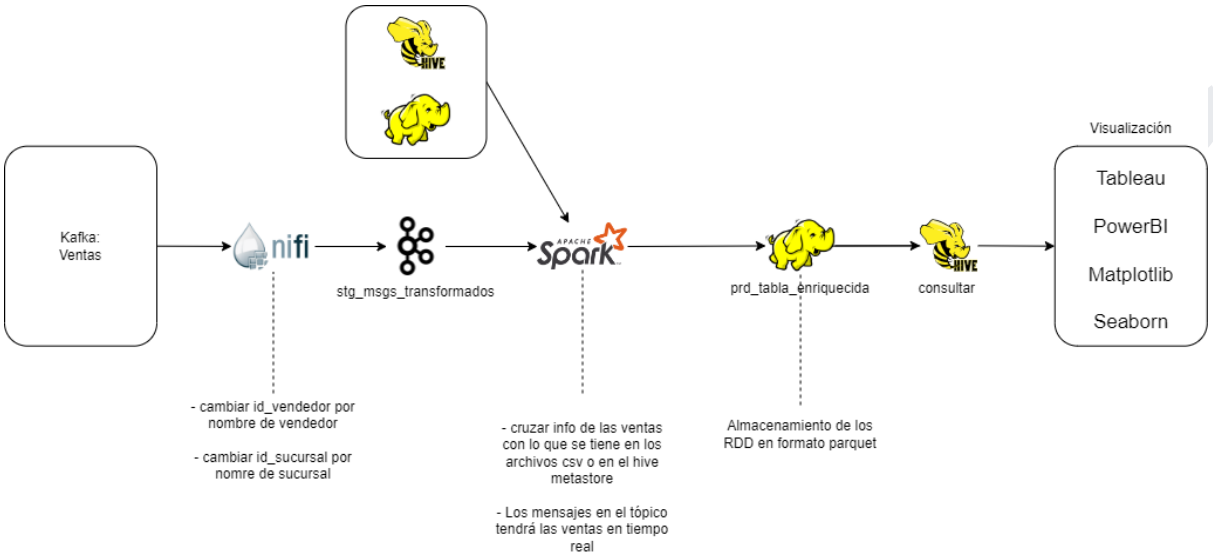
Se les proveerá de archivos en formato CSV que contendrán lo siguiente:

- **empleados.csv**: maestro de empleados
  - id_vendedor,sucursal,nombre
- **fact.csv**: tabla de hechos con compras
  - timestamp;sku;vendedor;cantidad
- **locales.csv**: maestro de sucursales
  - id_sucursal;nombre;tipo;
- **producto.csv**: maestro de productos
  - id_producto;familia;nombre;precio_unitario

La estructura de los mensajes que estarán disponibles en Kafka es la siguiente:

```
{
    “total”: <decimal>,
    “dinero_recibido”: <decimal>,
    “vuelto”: <decimal>,
    “vendedor”: <empleados.id_vendedor>,
    “sucursal”: <sucursal.id_sucursal>
}
```

Ejemplo de implementación conceptual:



La implementación específica de la solución es el desafío que enfrentarán.

- Existe un productor Kafka que está agregando datos de manera constante a un tópico.
- La ingesta de estos datos puede hacerse desde NiFi o Spark.
- El enriquecimiento lo pueden realizar mediante NiFi o Spark, ideal que incorporen la herramienta que aún no hayan aplicado.
- Los datos enriquecidos deberán estar disponibles para la consulta mediante SQL.
- Y posterior visualización con la herramienta que mejor manejen, siendo deseable que puedan armar algo con Flask y/o PowerBI.

Cuestiones a tener en cuenta:

- Se cuenta en el entorno corporativo con las diferentes herramientas que permitirán el acceso y la manipulación de los datos:

- Hadoop
- Kafka
  - Tópico con mensajes para esta consigna: `academia-kafka-ventas`
  - Brokers / Bootstrap servers: `dkafka01:9092,dkafka02:9092,dkafka03:9092`
- NiFi
  - URL del entorno: https://cdh001-e01.bancogalicia.com.ar:9444/nifi/?processGroupId=018111ac-d70b-1594-a5fb-48ff778256b8&componentIds=
  - Acceso: `legajo (con "L" en mayúscula) y contraseña de Windows`
- Spark
  - Ver configuración en notebook compartida en el módulo de la academia.

Anexo 1: Lab - SQL on Hadoop

Lab - SQL on Hadoop

Para el laboratorio vamos a utilizar HUE. A continuación, se detallan los pasos para conectarse al entorno de trabajo.

1. Desde citrix abrir un navegador web, por ejemplo, Google Chrome.
2. URL HUE: http://pcdp-e03.bancogalicia.com.ar:8889/
3. Ingresar usuario y clave de LDAP

Revisar y entender la estructura de los archivos

1. **Empleados.csv**: maestro de empleados
    a. id_vendedor,sucursal,nombre
2. **fact.csv**: tabla de hechos con compras
    a. timestamp;sku;vendedor;cantidad
3. **locales.csv**: maestro de sucursales
    a. id_sucursal;nombre;tipo;
4. **producto.csv**: maestro de productos
    a. id_producto;familia;nombre;precio_unitario
5. Para los joins considerar:
    a. SKU = id_producto
    b. id_vendedor = vendedor
    c. id_sucursal = sucursal

Data Lake o Pre-landing Zone

Desde una notebook llamada “Data Lake”

1. Crear **tablas externas** apuntando a cada archivo usando HQL definiendo la metadata.
2. Validar con una query estén accesibles todos los registros de cada uno de los archivos.

TABLA EXTERNA EMPLEADOS

```
CREATE EXTERNAL TABLE academia_lnd.empleados
(
  id_vendedor integer,
 sucursal integer,
 nombre string
)
        COMMENT 'Tabla que lee archivo de empleados'
        ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ','
        STORED AS TEXTFILE
                LOCATION
'hdfs://GALICIAHADOOP/galicia/d/landing_files/academia_de/empleados'
        TBLPROPERTIES ("skip.header.line.count"="1");
```

- TABLA EXTERNA EMPLEADOS

```
CREATE EXTERNAL TABLE academia_lnd.productos
(
 id_producto integer,
 familia string,
 nombre string,
 precio_unitario integer
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ';'
STORED AS TEXTFILE
LOCATION 'hdfs://GALICIAHADOOP/galicia/d/landing_files/academia_de/productos'
TBLPROPERTIES ("skip.header.line.count"="1");

LOAD DATA INPATH 'hdfs://GALICIAHADOOP/galicia/d/landing_files/academia_de/archivos/producto.csv' INTO TABLE d_lnd_tables.
```

- TABLA EXTERNA SUCURSALES

```
CREATE EXTERNAL TABLE academia_lnd.sucursales
(
 id_sucursal integer,
 nombre string,
 tipo string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ';'
STORED AS TEXTFILE
LOCATION 'hdfs://GALICIAHADOOP/galicia/d/landing_files/academia_de/sucursales'
TBLPROPERTIES ("skip.header.line.count"="1");
```

- TABLA EXTERNA FACT

```
CREATE EXTERNAL TABLE academia_lnd.fact
(
 fecha string,
 sku integer,
 vendedor integer,
 cantidad integer
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ';'
STORED AS TEXTFILE
LOCATION 'hdfs://GALICIAHADOOP/galicia/d/landing_files/academia_de/fact'
TBLPROPERTIES ("skip.header.line.count"="1");

LOAD DATA INPATH 'hdfs://GALICIAHADOOP/galicia/d/landing_files/academia_de/archivos/fact.csv' INTO TABLE d_lnd_tables.fact;
```

2 – VALIDACIÓN DE TABLAS

```
-- Tabla Empleados, se validan datos y cantidad
SELECT * FROM academia_lnd.empleados;
SELECT count(*) AS cantidad FROM academia_lnd.empleados;

-- Tabla productos, se validan datos y cantidad de productos por familia
SELECT * FROM academia_lnd.productos;
SELECT familia,count(*) AS cantidad
FROM academia_lnd.productos
WHERE familia='Helados'
GROUP BY familia
ORDER BY familia;

-- Tabla Sucursales, se validan datos y cantidad
SELECT * FROM academia_lnd.sucursales;
SELECT count(*) AS cantidad FROM academia_lnd.sucursales;

-- Tabla Fact, se validan datos, cantidad y tipo de tabla
SELECT * FROM academia_lnd.fact;
SELECT count(*) AS cantidad FROM academia_lnd.fact;
DESCRIBE EXTENDED academia_lnd.fact;
```

Data Sandbox

Desde una notebook llamada “Data Sandbox”.

1. Validar la integridad referencial de los datos (registros de hechos que no coincidan con dimensiones)
2. Crear una tabla **interna en formato AVRO** llamada:
    a. **fact_ventas**: donde estén los registros curados en términos integrales (es decir, que no haya registros con id’s inexistentes) y adicionalmente separar la fecha (campo timestamp) en 3 campos independientes: día, mes, año.
    b. Tomando la info de la tabla creada en la sección anterior.
3. Crear 2 tablas **internas en formato Parquet** (tomando la info de las tablas externas de la sección anterior) llamadas:
    a. **dim_vendedor**: que integre la información de vendedor con la de locales, generando una estructura con las siguientes columnas:
        i. id_vendedor
        ii. vendedor_nombre
        iii. sucursal_nombre
        iv. region_nombre
    b. **dim_producto**: respetando la estructura del archivo equivalente

1.1 - VALIDACIÓN Fact vs Empleados

```
-- Con la consulta de abajo vemos que hay 3422 registros de la tabla fact que referencian a registros inexistentes de la tabla empleados.
SELECT COUNT(*) AS sin_referencia
FROM academia_lnd.fact f
LEFT JOIN academia_lnd.empleados e ON f.vendedor = e.id_vendedor
WHERE e.id_vendedor IS NULL;
```

1.2 - VALIDACIÓN Fact vs Productos

```
-- Con la consulta de abajo vemos que hay 8436 registros de la tabla fact que referencian a registros inexistentes de la tabla productos.
SELECT COUNT(*) AS sin_referencia
FROM academia_lnd.fact f
LEFT JOIN academia_lnd.productos p ON f.sku = p.id_producto
WHERE p.id_producto IS NULL;
```

1.3 - VALIDACIÓN Empleados vs Sucursales

```
-- Con la consulta de abajo vemos que no hay registros de la tabla empleados que referencian a registros inexistentes de la tabla sucursales.
SELECT COUNT(*) AS sin_referencia
FROM academia_lnd.empleados e
LEFT JOIN academia_lnd.sucursales s ON e.sucursal = s.id_sucursal
WHERE s.id_sucursal IS NULL;
```

1.4 - VALIDACIÓN GENERAL

*-- Con la consulta de abajo vemos que en total de los **100.000** registros de la tabla fact, sólo **88.422** tienen integridad referencial correcta a las tablas de dimensiones. **11.578** registros son corruptos en integridad referencial.*

```
SELECT COUNT(*)
FROM academia_lnd.fact f
INNER JOIN academia_lnd.empleados e ON e.id_vendedor = f.vendedor
INNER JOIN academia_lnd.productos p ON p.id_producto = f.sku;
```

2 – ALTERNATIVA 1 – *CREATE TABLE AS*

```
CREATE TABLE academia_stg.fact_ventas
STORED AS AVRO AS
SELECT from_unixtime(unix_timestamp(fecha , 'dd/MM/yyyy')) as fecha,
   year(from_unixtime(unix_timestamp(fecha , 'dd/MM/yyyy'))) as ano,
   month(from_unixtime(unix_timestamp(fecha , 'dd/MM/yyyy'))) as mes,
   day(from_unixtime(unix_timestamp(fecha , 'dd/MM/yyyy'))) as dia,
   f.sku as id_producto,
   f.vendedor as id_vendedor,
   f.cantidad as cantidad
FROM academia_lnd.fact f
INNER JOIN academia_lnd.empleados e ON e.id_vendedor = f.vendedor
INNER JOIN academia_lnd.productos p ON p.id_producto = f.sku;
```

ALTERNATIVA 2

-- Primero creamos una vista con el formato correcto de las fechas y luego la utilizamos para crear la tabla

```
CREATE VIEW academia_views.fechas AS
SELECT DISTINCT fecha, from_unixtime(unix_timestamp(fecha , 'dd/MM/yyyy')) as fecha_date
    FROM academia_lnd.fact;

CREATE TABLE academia_stg.fact_ventas
STORED AS AVRO AS
SELECT fecha_date as fecha,
   year(fecha_date) as ano,
   month(fecha_date) as mes,
   day(fecha_date) as dia,
   f.sku as id_producto,
   f.vendedor as id_vendedor,
   f.cantidad as cantidad
FROM academia_lnd.fact f
INNER JOIN academia_lnd.empleados e ON e.id_vendedor = f.vendedor
INNER JOIN academia_lnd.productos p ON p.id_producto = f.sku
INNER JOIN academia_stg.fechas fec ON fec.fecha = f.fecha;
```

3.1 – DIM VENDEDORES

```
CREATE TABLE academia_dl.dim_vendedores
STORED AS PARQUET AS
SELECT e.id_vendedor as id_vendedor, e.nombre as vendedor_nombre, s.nombre as sucursal_nombre, s.tipo as sucursal_tipo
FROM academia_lnd.empleados e
INNER JOIN academia_lnd.sucursales s ON e.sucursal = s.id_sucursal;

SELECT * FROM academia_dl.dim_vendedores;
```

3.2 – DIM PRODUCTOS

```
CREATE TABLE academia_dl.dim_productos
STORED AS PARQUET AS
SELECT * FROM academia_lnd.productos;

SELECT * FROM academia_dl.dim_productos;
```

Data Warehouse o Consume Zone

Desde una notebook llamada “Consume Zone”:

1. Crear **una tabla en formato Parquet** que replique la información de la tabla avro fact_ventas_final pero:
    a. que esté particionada por el campo mes.
    b. incluya el campo “monto total” resolviendo el cálculo entre cantidad del producto vendido y su precio unitario (tomado de la dimensión creada en el punto anterior).
2. Truncar la partición de diciembre dado los datos están mal.
3. Desde una notebook llamadas “consultas analíticas”, resolver las siguientes consultas usando las tablas creadas en los puntos anteriores.
    a. Top-10 de sucursales según monto vendido, indicando el monto, ordenado de mayor a menor.
    b. top_3_productos_vend: listado ordenado de mayor a menor con los mejor vendedores, mostrando los 3 productos más vendidos por cada uno y sus montos correspondientes.
    c. peores 3 sucursales: identificar las peores sucursales
4. Borrar las tablas externas ¿Se borran los datos?

1.1 – TABLA PARTICIONADA

```
CREATE TABLE academia_dl.fact_ventas_final
(
   fecha STRING,
   ano INT,
   dia INT,
   id_producto INT,
   id_vendedor INT,
   cantidad INT,
   monto_total INT)
PARTITIONED BY (mes INT)
STORED AS PARQUET;
```

1.2 – INSERTAR DATOS SOBRE TABLA PARTICIONADA

```
--Primero hay que habilitar la inserción de datos de manera dinámica sobre tablas particionadas. Para mayor detalle ir al link
SET hive.exec.dynamic.partition = true;
SET hive.exec.dynamic.partition.mode = nonstrict;

-- Ahora ya podemos realizar el insert sobre la tabla
INSERT INTO academia_dl.fact_ventas_final  PARTITION(mes) SELECT * FROM (
SELECT
   f.fecha as fecha,
   f.ano as ano,
   f.dia as dia,
   f.id_producto as id_producto,
   f.id_vendedor as id_vendedor,
   f.cantidad as cantidad,
   f.cantidad * p.precio_unitario as monto_total,
   f.mes as mes
  FROM academia_stg.fact_ventas f
  INNER JOIN academia_dl.dim_productos p ON f.id_producto = p.id_producto) dual;
```

2 – TRUNCAR PARTICIÓN

```
ALTER TABLE academia_dl.fact_ventas_final DROP IF EXISTS PARTITION (mes=12);
```

3.1 – CONSULTAS ANALÍTICAS – Top 10

```
SELECT sucursal_nombre, SUM(monto_total) as Monto_vendido
FROM academia_dl.fact_ventas_final f
INNER JOIN academia_dl.dim_vendedores v ON f.id_vendedor = v.id_vendedor
GROUP BY sucursal_nombre
ORDER BY SUM(monto_total) DESC
LIMIT 10;
```

3.2 – CONSULTAS ANALÍTICAS – Top 3 Productos Vendidos

```
SELECT *
FROM (
SELECT id_vendedor,
    vendedor_nombre,
    id_producto,
    nombre_producto,
    monto_vendido_por_producto,
    sum(monto_vendido_por_producto) OVER (PARTITION BY id_vendedor ORDER BY monto_vendido_por_producto DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) as total_vendido,
    rank() OVER (PARTITION BY id_vendedor ORDER BY monto_vendido_por_producto DESC) as rnk
  FROM(SELECT v.id_vendedor as id_vendedor,
      v.vendedor_nombre as vendedor_nombre,
      p.id_producto as id_producto,
      p.nombre as nombre_producto,
      SUM(monto_total) as monto_vendido_por_producto
    FROM academia_dl.fact_ventas_final f
    INNER JOIN academia_stg.dim_productos p ON p.id_producto = f.id_producto
    INNER JOIN academia_stg.dim_vendedores v ON v.id_vendedor = f.id_vendedor
    GROUP BY v.id_vendedor, v.vendedor_nombre, p.id_producto, p.nombre) tmp) final
WHERE final.rnk < 4
ORDER BY final.total_vendido DESC, final.rnk ASC;
```

3.3 – CONSULTAS ANALÍTICAS – Top 3 Peores sucursales

```
SELECT sucursal_nombre, SUM(monto_total) as monto_vendido
FROM academia_dl.fact_ventas_final f
INNER JOIN academia_dl.dim_vendedores v ON v.id_vendedor = f.id_vendedor
GROUP BY sucursal_nombre
ORDER BY monto_vendido
LIMIT 3;
```

4 – BORRADO DE TABLAS EXTERNAS

```
DROP TABLE academia_lnd.empleados;
DROP TABLE academia_lnd.productos;
DROP TABLE academia_lnd.sucursales;
DROP TABLE academia_lnd.fact;

-- Al ser tablas externas solamente se elimina la metadata y los datos persisten en el storage, en nuetro caso en HDFS
```

Anexo 2: Lab - Spark

Lab - Spark

Para el laboratorio vamos a utilizar Visual Studio Code.

Trabajaremos con los archivos del módulo 4

1. **Empleados.csv**: maestro de empleados
    a. id_vendedor,sucursal,nombre
2. **fact.csv**: tabla de hechos con compras
    a. timestamp;sku;vendedor;cantidad
3. **locales.csv**: maestro de sucursales
    a. id_sucursal;nombre;tipo;
4. **producto.csv**: maestro de productos
    a. id_producto;familia;nombre;precio_unitario
5. Para los joins considerar:
    a. SKU = id_producto
    b. id_vendedor = vendedor
    c. id_sucursal = sucursal

Ejercicio 1 – Análisis exploratorio

1. Leer los archivos de modo local.
2. Para cada uno:
    a. Mostrar 10 registros.
    b. Mostrar el total de registros.
3. Leer las tablas creadas en el módulo 4.
4. Comparar la cantidad de registros de los dataframes con las tablas.
5. Comparar los tipos de datos de las columnas entre dataframes y tablas.

Ejercicio 2 – Ingeniería de datos

1. Crear una tabla **interna en formato Parquet** llamada:
    a. **fact_ventas_spark**: donde estén los registros curados en términos integrales (es decir, que no haya registros con id's inexistentes) y adicionalmente separar la fecha (campo timestamp) en 3 campos independientes: día, mes, año.
1. Crear **una tabla externa fact_ventas_final_spark en formato Parquet** que replique la información de la tabla anterior, pero:
    a. que esté particionada por los campos año y mes.
    b. incluya el campo "monto" resolviendo el cálculo entre cantidad del producto vendido y su precio unitario (tomado de la dimensión creada en el módulo 4).
2. Escribir en HDFS, en formato Parquet el resultado de las siguientes consultas:
    a. Cantidad y promedio del precio de productos vendidos por sucursal.
    b. Cantidad de vendedores por sucursal para los productos cuyo precio promedio de venta sea mayor a 140.
3. Borrar las tablas externas ¿Se borran los datos?

Ejercicio 3 – Particiones de Spark

1. Chequear la salida en HDFS del ejercicio anterior. ¿Cuántos archivos parquet hay?
2. Utilizar la función coalesce para que spark guarde sólo 2 archivos.