Big Data Systems

Assignment, 2019

Big Data Architecture, HDFS and Hive

George Vogiatzis

P2821827

Table of Contents

Assignment Overview 2
- General Instructions 2
- Useful Software 2
- Additional Material 2
- Purpose of the assignment 2

Part 1: Theoretical questions 3
Part 2: Data Warehousing using Hive 7
- Introduction 7
- Questions/Tasks 8

Assignment Overview

Areas covered:

- Big Data Architecture.
- Hadoop Distributed File System (HDFS).
- Apache Hive.

Submission Deadline: 20/05/2019 00:00

General Instructions

The current assignment includes two parts of mandatory questions and tasks of 120 points, in total. The highest score that can be achieved in this assignment is 100 points (i.e., 20 points are considered as bonus points).

Note that although a certain dataset is provided and can be used to answer/complete the question/tasks of this assignment, it is not required to successfully complete the assignment. The questions could be answered even without using the dataset. However, it is recommended to use it (i.e., try to build the code requested over the given dataset, where it is feasible) in order to understand better what is asked, as well as practice over the areas considered in this assignment. The dataset is only used for practicing in Part 2.

Please provide your answers into a document (e.g., answers could be provided in the corresponding "answer" sections). Note that in case the answers are provided into a different document, please add the corresponding question-numbers in order to be clear which is the question of each answer.

Useful Software

Use the Cloudera Quickstart (CDH, for short) to practice on the tasks of this assignment (for installation details see slides in SWInstallation.pdf uploaded in the space of the course). Note that the services (from CDH) that could be used for completing this assignment are limited to the following:

- Apache Hadoop (including MapReduce and HDFS),
- Apache Hive, and
- Hue

Additional Material

- The given dataset (orders_dataset.zip) includes the following files:
  - order_details.csv: Details about orders.
  - orders.csv: Information about orders.
  - rates.zip: historical conversion rates to EUR.

Purpose of the assignment

The purpose of this assignment is to:

1. Understand the main concepts and approaches discussed during the corresponding lectures (big data architectures, HDFS, storage formats and Hive), and
2. Learn/practice on building and utilizing a star schema using Apache Hive and HDFS.

Part 1: Theoretical questions

<40 points>

Q1.1 <4pt.>: Which of the following issues may be caused by lot of small files in HDFS:

a) NameNode memory usage increases significantly.

b) Overall network load increases.

c) In MapReduce, the number of map tasks that need to process the same amount of data will be larger.

d) I/O rate will be faster.

Select the correct options (one or more) ***and explain*** your answer, **shortly**.

Answer: a, b, c

Explanation:

Every file, directory and block in HDFS is represented as an object in the namenode's memory, each of which occupies 150 bytes, as a rule of thumb. So a lot of small files, 10 million for example, each using a block, would use about 3 gigabytes of memory.

Network load will increase also as the number of checks with datanodes is proportional to number of blocks.

Map tasks usually process a block of input at a time (using the default FileInputFormat). If the file is very small and there are a lot of them, then each map task processes very little input, and there are a lot more map tasks, each of which imposes extra bookkeeping overhead. Compare a 1GB file broken into 16 64MB blocks, and 10,000 or so 100KB files. The 10,000 files use one map each, and the job time can be tens or hundreds of times slower than the equivalent one with a single input file.

Q1.2 <4pt.>: How is uniform data distribution across the servers achieved in HDFS?

a) By splitting files into blocks

b) By replication

Select one of the aforementioned options ***and explain*** your answer, **shortly**.

Answer: a, b

HDFS data blocks might not always be placed uniformly across data nodes, meaning that the used space for one or more data nodes can be underutilized. Therefore, HDFS supports rebalancing data blocks using various models. One model might move data blocks from one data node to another automatically if the free space on a data node falls too low. Another model might dynamically create additional replicas and rebalance other data blocks in a cluster if a sudden increase in demand for a given file occurs.

Q1.3 <4pt.>: If you have a very important file and you aim to store it into HDFS, what is the best way to minimize the risk from losing it?

Answer:A common way of avoiding data loss is through replication. When files are being written the data nodes form a pipeline to write the replicas in sequence. Data is sent through the pipeline in packets (smaller

than a block), each of which must be acknowledged to count as a successful write. If a data node fails while the block is being written, it is removed from the pipeline. When the current block has been written, the name node will re-replicate it to make up for the missing replica due to the failed data node. Subsequent blocks will be written using a new pipeline with the required number of datanodes.

Q1.4 <4pt.>: You were told that two servers in HDFS were down: Datanode and Namenode. Which would be your reaction:

a) It's OK, replication factor is 3.

b) Restore Datanode first.

c) Restore Namenode first.

Select one of the aforementioned options ***and explain*** your answer, **shortly**.

Answer: c

Namenode , also called master node, is the main and heartbeat node of HDFS. It stores the metadata in RAM for quick access and tracking of the files across HADOOP cluster. If Namenode is down the whole HDFS is inaccessible so Namenode is going to be the first to be restored.

Q1.5 <4pt.>: You are writing a 10GB file into HDFS with a replication of 2 and block size of 64MB. How much **total** disk space will this file use? **E*xplain*** your answer, **shortly**.

Answer: 20096MBytes

Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block's storage (10GB/64MB=156,25 blocks), therefore the correct answer is file size * replication factor=20GB

Q1.6 <4pt.>: Your colleague mistakenly dropped the table 'access_log' created with the following statement, in Hive:

```
CREATE EXTERNAL TABLE access_log (
id STRING,
user_id STRING,
request STRING,
response STRING,
status_code INT
 ) LOCATION "/data/access_logs"
```

Which would be your reaction? **Explain** your answer**, shortly**.

Answer: I would just create an external table again (same schema)

Dropping an external table in hive does not remove data, so in the above case we just create an external table with the same shcema and set the LOCATION pointing to the path where our data reside.

Q1.7 <4pt.>: You have a bunch of data in your local filesystem (which is **not** part of the cluster) and you need to load it in Hive. Describe the steps that you are going to follow in order the data to be accessible through Hive queries. *Code is not required, just a short description of the steps.*

Answer:

(1) Create external delta tables to read the input files. For validating the data, we create external tables by applying string as a data type.

(2) Create views to convert data types if necessary.

(3) Load the data into final table.

Q1.8 <4pt.>: You have an external table in Hive and want to store this data in more compact and efficient format. Your decision is:

a) Create a table in a new format with the CREATE TABLE statement and fill it from the external table with the INSERT INTO TABLE statement.

b) CREATE VIEW over an external table.

c) Create and fill a new table with the CREATE TABLE ... AS SELECT statement.

Select one of the options ***and explain*** your answer, **shortly**.

In addition, provide at least one type of format that could improve the performance on query aggregations. **E*xplain*** your answer, **shortly**.

Answer: b

Views & storage are not associated. A view is purely a logical construct (an alias for a query) with no physical data behind it and views send map reduce queries each time they are called. So when creating a view no additional storage space is required.

Parquet format can improve performance on query aggregations. It is a column oriented format so it brings the benefit of improved performance and better compression

Q1.9 <4pt.>: Why does partitioning optimize Hive queries? Please provide a **short** answer.

Answer:

Basically, by partitioning all the entries for the various columns of the dataset are segregated and stored in their respective partition. Hence, while we write the query to fetch the values from the table, only the required partitions of the table are queried. Thus, it reduces the time taken by the query to yield the result.

Q1.10 <4pt.>: You join two Hive tables: A(key INT, value STRING) and B(key INT, value STRING). Table B is small enough to be stored in RAM of a single compute node in the cluster and A is much bigger than B (A exceeds the average RAM of a cluster node). Provide an efficient query optimizing the join (INNER JOIN over A.key=B.key) between A and B, in Hive. Assume that the query returns A.value and B.value.

Answer:

```
SELECT
/*+ MAPJOIN(B) */ A.value, B.value
FROM A
CROSS JOIN B
WHERE A.key = B.key;
```

Part 2: Data Warehousing using Hive

<80 points>

Introduction

The purpose of this part is to apply transformations and analysis tasks over the data received through files. We assume that you receive data related to orders of an international company with multiple offices. Based on this data, you initially aim to structure the data into a simple star schema, and then apply further analysis. Please find below details about the files format and data considered as input.

Main Orders File (file name: orders.csv):

Field	Description
ORDER_NUMBER	Order unique identifier
ORDER_DATE	Date of Order
SHIPPED_DATE	Date the order shipped
STATUS	Order status
COMMENTS	Comments over the order
CUSTOMER_NUMBER	Customer unique identifier
CUSTOMER_NAME	Customer name
CUST_CITY	City of customer
CUST_STATE	State of customer
CUST_COUNTRY	Country of customer
CUST_COUNTRY_ISO	ISO code of customer country
SALES_CURRENCY	Currency of sales price
SALES_REP_ID	Sales representative id
SALES_REP_FIRSTNAME	First name of Sales representative
SALES_REP_LASTNAME	Surname of Sales representative
OFFICE_CODE	Code of office the sales representative belongs to
REPORTING_PATH	Sales representative reporting path
OFFICE_CITY	City of office
OFFICE_STATE	State of office
OFFICE_TERRITORY	Territory of office
OFFICE_COUNTRY	Office country

Order Details (file name: order_details.csv):

Field	Description
ORDER_NUMBER	Order unique identifier
PRODUCT_CODE	Unique identifier of the product

PRODUCT_NAME	Name of the product
PRODUCT_CATEGORY	Category of the product
PRODUCT_VENDOR	Vendor of the product
QUANTITY_IN_STOCK	Quantity in stock
BUY_PRICE	Purchase price, per item
QUANTITY_ORDERED	Number of items ordered, for the specific product
UNIT_PRICE	Price of the product (per item)

Note the following:

- The Sales currency included in the orders file applies to all the corresponding sales prices (UNIT_PRICE) in the order_details file. We consider that the currency of BUY_PRICE is always EUR.
- REPORTING_PATH refers to organizational structure of the sales department. i.e., the SALES_REP_ID of the line manager is the first element of the list, his/her line manager is included as second element, etc. The last element of the list is always -1, indicating that there is no line manager of the head of department.

Currency conversion rates (from any currency to EUR):

- Files included in zip file rates.zip.
- Each file includes the conversion rates for each date, in JSON format.
- The date included in both the file name and an element with key "date".
- All the rates included in sub-element named "rates".

Questions/Tasks

Q2.1 <7pt.>: Provide the statements in order to create the following folder structure within the HDFS folder "*cloudera*" located in the path */user/cloudera.*

- RawData within the cloudera folder.
- OrdersData in RawData folder.
- Rates in RawData folder.

Answer:

(hdfs/terminal)

hdfs dfs -mkdir /user/cloudera/RawData

hdfs dfs -mkdir /user/cloudera/RawData/OrdersData

hdfs dfs -mkdir /user/cloudera/RawData/Rates

Q2.2 <7pt.>: Provide the statements in order to load the following data into the corresponding HDFS folder.

Data type	**HDFS Location**
orders.csv	/user/cloudera/RawData/OrdersData
order_details.csv	/user/cloudera/RawData/OrdersData

Exchange rates	/user/cloudera/RawData/Rates

Consider that the files have already been uploaded into a node having HDFS client set-up properly.

Answer:

Execute following commands in terminal:

hdfs dfs -put ./orders.csv /user/cloudera/RawData/OrdersData

hdfs dfs -put ./order_details.csv /user/cloudera/RawData/OrdersData

hdfs dfs -put ./rates/*.json /user/cloudera/RawData/Rates

Q2.3 <8pt.>: Provide the statements in order to make the following data accessible through Hive (i.e., the data could be queried through Hive).

- orders.csv
- order_details.csv
- Exchange rates

Creation statement of the corresponding tables and "loading" statements are requested. **In addition,** initially, create a new database with name "order_data" and then create all the tables in this database.

Hints:

- *There are multiple approaches/statements to load the data; each of them is accepted as answers.*
- *Use a complex data type for defining the column REPORTING_PATH.*
- *Identify and define correctly field and collection delimiters.*
- *Do not forget to ignore the headers.*

Answer:

Hue Editor:

CREATE DATABASE IF NOT EXISTS order_data;

USE order_data;

CREATE TABLE IF NOT EXISTS orders (

ORDER_NUMBER STRING,

ORDER_DATE STRING,

SHIPPED_DATE STRING,

STATUS STRING,

COMMENTS STRING,

CUSTOMER_NUMBER STRING,

```
CUSTOMER_NAME STRING,
CUST_CITY STRING,
CUST_STATE STRING,
CUST_COUNTRY STRING,
CUST_COUNTRY_ISO STRING,
SALES_CURRENCY STRING,
SALES_REP_ID  STRING,
SALES_REP_FIRSTNAME STRING,
SALES_REP_LASTNAME  STRING,
OFFICE_CODE  STRING,
REPORTING_PATH MAP<STRING,ARRAY<STRING>>,
OFFICE_CITY STRING,
OFFICE_STATE  STRING,
OFFICE_TERRITORY STRING,
OFFICE_COUNTRY STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION '/user/cloudera/RawData/OrdersData/order'
tblproperties ("skip.header.line.count"="1");

CREATE TABLE IF NOT EXISTS order_details (
ORDER_NUMBER STRING,
PRODUCT_CODE STRING,
PRODUCT_NAME STRING,
PRODUCT_CATEGORY STRING,
PRODUCT_VENDOR STRING,
QUANTITY_IN_STOCK STRING,
BUY_PRICE STRING,
QUANTITY_ORDERED STRING,
```

```
UNIT_PRICE STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION '/user/cloudera/RawData/OrdersData/order_details'
tblproperties ("skip.header.line.count"="1")

CREATE TABLE rates (
rates string
)
STORED AS TEXTFILE
LOCATION '/user/cloudera/RawData/Rates'
```

Terminal(load the data):

```
hdfs dfs -cp /user/cloudera/RawData/OrdersData/orders.csv /user/cloudera/RawData/OrdersData/order
hdfs dfs -cp /user/cloudera/RawData/OrdersData/order_details.csv /user/cloudera/RawData/OrdersData/order_details
hdfs dfs -cp /user/cloudera/RawData/Rates/*.json /user/cloudera/RawData/Rates/rates
```

Q2.4 <7pt.>: Provide the statements in order to create a partitioned table named “stg_orders” for storing historical orders data. Select proper partition columns.

Answer:

```
create view delta_orders_view
as
SELECT
ORDER_NUMBER,
cast(to_date(from_unixtime(unix_timestamp(ORDER_DATE, 'yy-MM-dd HH:mm'))) as date) as ORDER_DATE,
month(cast(to_date(from_unixtime(unix_timestamp(ORDER_DATE, 'yy-MM-dd HH:mm'))) as date)) as order_month,
year(cast(to_date(from_unixtime(unix_timestamp(ORDER_DATE, 'yy-MM-dd HH:mm'))) as date)) as order_year,
cast(to_date(from_unixtime(unix_timestamp(SHIP_DATE, 'yy-MM-dd HH:mm'))) as date) as SHIP_DATE,
```

```
STATUS,
COMMENTS,
CUSTOMER_NUMBER,
CUSTOMER_NAME,
CUST_CITY,
CUST_STATE,
CUST_COUNTRY,
CUST_COUNTRY_ISO,
SALES_CURRENCY,
SALES_REP_ID,
SALES_REP_FIRSTNAME,
SALES_REP_LASTNAME,
OFFICE_CODE,
REPORTING_PATH,
OFFICE_CITY,
OFFICE_STATE,
OFFICE_TERRITORY,
OFFICE_COUNTRY
FROM orders;

CREATE TABLE IF NOT EXISTS stg_orders (
ORDER_NUMBER STRING,
ORDER_DATE DATE,
SHIPPED_DATE DATE,
STATUS STRING,
COMMENTS STRING,
CUSTOMER_NUMBER STRING,
CUSTOMER_NAME STRING,
CUST_CITY STRING,
CUST_STATE STRING,
CUST_COUNTRY STRING,
```

```
CUST_COUNTRY_ISO STRING,
SALES_CURRENCY STRING,
SALES_REP_ID  STRING,
SALES_REP_FIRSTNAME STRING,
SALES_REP_LASTNAME  STRING,
OFFICE_CODE  STRING,
REPORTING_PATH MAP<STRING,ARRAY<STRING>>,
OFFICE_CITY STRING,
OFFICE_STATE  STRING,
OFFICE_TERRITORY STRING,
OFFICE_COUNTRY STRING
)
PARTITIONED BY (order_year int, order_month int)
STORED AS orc;

set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
INSERT INTO stg_orders
PARTITION (order_year, order_month)
SELECT
ORDER_NUMBER,
ORDER_DATE,
SHIPPED_DATE,
STATUS,
COMMENTS,
CUSTOMER_NUMBER,
CUSTOMER_NAME,
CUST_CITY,
CUST_STATE,
CUST_COUNTRY,
CUST_COUNTRY_ISO,
```

SALES_CURRENCY,

SALES_REP_ID,

SALES_REP_FIRSTNAME,

SALES_REP_LASTNAME,

OFFICE_CODE,

REPORTING_PATH,

OFFICE_CITY,

OFFICE_STATE,

OFFICE_TERRITORY,

OFFICE_COUNTRY,

order_year,

order_month

FROM delta_orders_view;

Q2.5 <7pt.>: Consider the following staging tables storing the historical, raw records for order_details and rates, respectively:

```
CREATE TABLE IF NOT EXISTS order_data.stg_order_details (
ORDER_NUMBER BIGINT,
PRODUCT_CODE STRING,
PRODUCT_NAME STRING,
PRODUCT_CATEGORY STRING,
PRODUCT_VENDOR STRING,
QUANTITY_IN_STOCK INT,
BUY_PRICE DECIMAL(15,2),
QUANTITY_ORDERED INT,
UNIT_PRICE DECIMAL(15,2)
)
STORED AS orc;
```

```
CREATE TABLE IF NOT EXISTS order_data.stg_conv_rates (
  INDATE DATE,
  RATES STRING
)
STORED AS orc;
```

Consider also the external tables, ext_order_details and ext_rates, over the corresponding raw data files, where all the columns are given in STRING data type. Provide the statements loading the data from the external tables to the stg_ tables by converting the columns into the proper data type.

Hint:

- *Consider that the external table ext_rates has a single column.*
- *Consider that each record of the external table ext_rates has all the rates of a certain date.*

Answer:

```
INSERT INTO order_data.stg_order_details(
CAST(ORDER_NUMBER AS BIGINT) AS ORDER_NUMBER,
PRODUCT_CODE ,
PRODUCT_NAME ,
PRODUCT_CATEGORY ,
```

```
PRODUCT_VENDOR ,
CAST(QUANTITY_IN_STOCK AS INT) AS QUANTITY_IN_STOCK,
CAST(BUY_PRICE AS DECIMAL(15,2)) AS BUY_PRICE,
CAST(QUANTITY_ORDERED AS INT) AS QUANTITY ORDERER,
CAST(UNIT_PRICE AS DECIMAL(15,2)) AS UNIT_PRICE
)
FROM ext_order_details ;

INSERT INTO order_data.stg_conv_rates(
CAST(GET_JSON_OBJECT(Rates,'$.date') AS DATE) AS INDATE
GET_JSON_OBJECT(RATES,'$.rates') AS RATES
)
FROM ext_rates
```

Q2.6 <7pt.>: Provide the statements in order to build the surrogate keys of the dimensions dim_status and dim_product, where their definitions are given as follows:

```
CREATE TABLE order_data.dim_status (
   STATUS_KEY INT,
   STATUS STRING
)
STORED AS orc;
```

```
CREATE TABLE order_data.dim_product (
   PRODUCT_KEY INT,
   PRODUCT_CODE STRING,
   PRODUCT_NAME STRING,
   PRODUCT_CATEGORY STRING,
   PRODUCT_VENDOR STRING
)
STORED AS orc;
```

Use the following table in order to store the surrogate keys:

```
CREATE TABLE IF NOT EXISTS order_data.dim_keys (
  dim_key INT,
  dim_type STRING,
  value STRING,
  created_at TIMESTAMP
)
STORED AS orc;
```

Hint:

- *Provide the queries (including insertion clauses) over the tables* order_data.stg_order_details and order_data.stg_orders defined in previous tasks.

Answer:

```
INSERT INTO order_data.dim_keys

(dim_key, dim_type, value, created_at)

select

row_number() over () as dim_key,

dim_type,

value,

created_at

from (
```

```
select
'product' as dim_type,
 product as value,
 current_timestamp() as created_at
FROM order_data.stg_order_details
group by product) m
order by value;

INSERT INTO order_data.dim_keys
(dim_key, dim_type, value, created_at)
select
row_number() over () as dim_key,
dim_type,
value,
created_at
from (
select
'status' as dim_type,
 status as value,
 current_timestamp() as created_at
FROM order_data.stg_orders
group by status) m
order by value;
```

Q2.7 <7pt.>: Provide the statements in order to insert data into the following dimensions.

```
CREATE TABLE order_data.dim_customer
(
   CUST_KEY BIGINT,
   CUSTOMER_NUMBER INT,
   CUSTOMER_NAME STRING,
   CUST_CITY STRING,
   CUST_STATE STRING,
   CUST_COUNTRY STRING,
   CUST_COUNTRY_ISO STRING
)
STORED AS orc;
```

```
CREATE TABLE order_data.dim_sales_rep
(
   SALES_REP_KEY INT,
   SALES_REP_ID INT,
   SALES_REP_FIRSTNAME STRING,
   SALES_REP_LASTNAME STRING,
   OFFICE_CODE STRING,
   OFFICE_CITY STRING,
   OFFICE_STATE STRING,
   OFFICE_TERRITORY STRING,
   OFFICE_COUNTRY STRING
)
STORED AS orc;
```

Use the tables order_data.stg_order_details and order_data.stg_orders defined in previous tasks in order to retrieve the dimension data. Note that the surrogate keys CUST_KEY and SALES_REP_KEY are defined in the following table (column dim_key) for each distinct value of CUSTOMER_NUMBER and SALES_REP_ID, respectively, in the corresponding columns in order_data.stg_order_details and order_data.stg_orders. The surrogate keys are stored in the column "value". The dim_type for CUST_KEY is "customer", while the dim_type for SALES_REP_ID is "salesrep".

```
CREATE TABLE IF NOT EXISTS order_data.dim_keys (
  dim_key int,
  dim_type string,
  value string,
  created_at TIMESTAMP
)
STORED AS orc;
```

Hint:

- *Notice that* CUSTOMER_NUMBER and SALES_REP_ID are of type INT, while they are stored in column "value" of dim_keys. Hence, conversion of data types is required.

Answer:

ALTER TABLE stg_orders CHANGE CUSTOMER_NUMBER CUSTOMER_NUMBER INT AFTER comments ;

ALTER TABLE stg_orders CHANGE sales_rep_id sales_rep_id INT AFTER sales_currency ;

INSERT INTO dim_customer

(CUST_KEY, CUSTOMER_NUMBER, CUSTOMER_NAME, CUST_CITY, CUST_STATE, CUST_COUNTRY, CUST_COUNTRY_ISO)

SELECT k.dim_key,

t.CUSTOMER_NUMBER, t.CUSTOMER_NAME, t.CUST_CITY, t.CUST_STATE, t.CUST_COUNTRY, t.CUST_COUNTRY_ISO

FROM dim_keys k

join (select t.CUSTOMER_NUMBER, t.CUSTOMER_NAME, t.CUST_CITY, t.CUST_STATE, t.CUST_COUNTRY, t.CUST_COUNTRY_ISO

    from stg_orders t

group by t.CUSTOMER_NUMBER, t.CUSTOMER_NAME, t.CUST_CITY, t.CUST_STATE, t.CUST_COUNTRY, t.CUST_COUNTRY_ISO) t

on k.dim_type='customer' and k.value=t.CUSTOMER_NUMBER;

INSERT INTO dim_sales_rep

(SALES_REP_KEY, SALES_REP_ID, SALES_REP_FIRSTNAME, SALES_REP_LASTNAME, OFFICE_CODE, OFFICE_CITY, OFFICE_STATE, OFFICE_TERRITORY, OFFICE_COUNTRY)

SELECT k.dim_key,

t.SALES_REP_ID, t.SALES_REP_FIRSTNAME, t.SALES_REP_LASTNAME, t.OFFICE_CODE, t.OFFICE_CITY, t.OFFICE_STATE, t.OFFICE_TERRITORY, t.OFFICE_COUNTRY

FROM dim_keys k

join (select t.SALES_REP_ID, t.SALES_REP_FIRSTNAME, t.SALES_REP_LASTNAME,, t.OFFICE_CODE, t.OFFICE_CITY, t.OFFICE_STATE, t.OFFICE_TERRITORY, t.OFFICE_COUNTRY

from stg_orders t

group by t.SALES_REP_ID, t.SALES_REP_FIRSTNAME, t.SALES_REP_LASTNAME,, t.OFFICE_CODE, t.OFFICE_CITY, t.OFFICE_STATE, t.OFFICE_TERRITORY, t.OFFICE_COUNTRY) t

on k.dim_type='salesrep' and k.value=t.SALES_REP_ID;

Q2.8 <7pt.>: Provide the statements in order to create the following time/date dimension.

```
CREATE TABLE IF NOT EXISTS order_data.dim_date (
  date_id INT,
  date DATE,
  day_of_week INT,
  year INT,
  month INT,
  week_of_year INT
)
STORED AS orc;
```

Hint:

- *Use the functions posexplode, split, repeat and datediff.*
- *Find the minimum date included either in ORDER_DATE or in SHIPPED_DATE of the table stg_orders.*
- *Date functions: link.*

Answer:

-- The following expression initializes the time/date time by generating all the dates between the minimum date included in stg_transaction_headers and the current date.

with min_max_dates as (

select min(order_date) as min_order_date,current_date() as today

from order_data.stg_order_details

)

INSERT INTO order_data.dim_date

(date_id, `date`,'day_of_week', `year`, `month`, week_of_year)

select

row_number() over () as date_id,

date,

dayofweek(date) as day_of_week

year(date) as year,

month(date) as month,

weekofyear(date) as week_of_year

from (select

date_add(t.min_order_date, a.pos) as date

from (

select posexplode(split(repeat("o", datediff(today, t.min_order_date)), "o"))

from min_max_dates t

) a, min_max_dates t) d

order by date

Q2.9 <8pt.>: Provide the statements in order to build the following fact table over the data stored in the tables order_data.stg_order_details and order_data.stg_orders. Use also the following dimensions defined in the previous tasks:

- order_data.dim_status
- order_data.dim_product
- order_data.dim_customer
- order_data.dim_sales_rep
- order_data.dim_date

Definition of fact table:

```
CREATE TABLE order_data.fact_orders (
   ORDER_DATE_KEY INT, -- Surrogate key of time dimension for ORDER_DATE
   SHIPPED_DATE_KEY INT, -- Surrogate key of time dimension for SHIPPED_DATE
   CUST_KEY BIGINT, -- Surrogate key of customer dimension for CUSTOMER_NUMBER
   STATUS_KEY INT, -- Surrogate key of status dimension for STATUS
   PRODUCT_KEY INT, -- Surrogate key of product dimension for PRODUCT_CODE
   SALES_REP_KEY INT, -- Surrogate key of sales_rep dimension for SALES_REP_ID
   NUMBER_OF_ORDERS INT, -- Number of distinct orders
   TOTAL_QUANTITY_ORDERED INT, -- Total number of QUANTITY_ORDERED
   SALES_AMOUNT_IN_EUR DECIMAL(15,2), -- Total amount paid by the customer in EUR
   MAX_UNIT_PRICE_IN_EUR DECIMAL(15,2), -- Maximum unit price in EUR
   AVG_UNIT_PRICE_IN_EUR DECIMAL(15,2), -- Average unit price in EUR
   MIN_UNIT_PRICE_IN_EUR DECIMAL(15,2), -- Minimum unit price in EUR
   TOTAL_PURCHASE_AMOUNT_IN_EUR DECIMAL(15,2) -- Total amount paid by the company in EUR
)
STORED AS orc;
```

Hints:

- *The measures are given for each unique combination of keys.*
- *The calculation of measures SALES_AMOUNT_IN_EUR, MAX_UNIT_PRICE_IN_EUR, AVG_UNIT_PRICE_IN_EUR, MIN_UNIT_PRICE_IN_EUR is based on QUANTITY_ORDERED, UNIT_PRICE and the corresponding conversion rates.*
- *The calculation of the measure TOTAL_PURCHASE_AMOUNT_IN_EUR is based on the QUANTITY_ORDERED and BUY_PRICE.*
- *Both ORDER_DATE_KEY INT, and SHIPPED_DATE_KEY get values from the dimension table order_data.dim_date.*

Answer:

```
CREATE TABLE tmp_combined_orders (
   ORDER_NUMBER BIGINT,
   ORDER_DATE DATE,
   CUSTOMER_NUMBER BIGINT,
   SHIPPED_DATE DATE,
   PRODUCT_CODE BIGINT,
   STATUS STRING,
   SALES_CURRENCY STRING,
   SALES_REP_ID BIGINT,
   QUANTITY_ORDERED INT,
   BUY_PRICE DECIMAL(15,3),
   UNIT_PRICE DECIMAL(15,3)
   rate_to_eur decimal(15,3)
)
STORED AS orc;


-- For each record, we extract from conversion rates json the corresponding rate.
insert into order_data.tmp_combined_orders
(ORDER_NUMBER, ORDER_DATE, CUSTOMER_NUMBER, SHIPPED_DATE, PRODUCT_CODE, STATUS, SALES_CURRENCY, SALES_REP_ID, QUANTITY_ORDERED, BUY_PRICE, UNIT_PRICE, rate_to_eur)
select
d.ORDER_NUMBER,
```

```
o.ORDER_DATE,
o.CUSTOMER_NUMBER
o.SHIPPED_DATE,
d.PRODUCT_CODE,
o.STATUS,
o.SALES_CURRENCY,
o.SALES_REP_ID,
d.QUANTITY_ORDERED,
d.BUY_PRICE
d.UNIT_PRICE,
coalesce(cast(get_json_object(c.rates,concat('$.',h.currency)) as decimal),1) as rate_to_eur
from stg_orders o
join stg_order_details d
on o.ORDER_NUMBER=d.ORDER_NUMBER
join ext_rates_view c
on c.ORDER_DATE=o.ORDER_DATE;
```

```
INSERT INTO fact_orders (ORDER_DATE, SHIPPED_DATE, CUST_KEY, STATUS_KEY,
PRODUCT_KEY, SALES_REP_KEY, number of orders, total quantity ordered, sales amount in EUR, max
unit price in EUR, avg unit price in EUR, min unit price in EUR, total purchase amount in EUR)
select
  date_id,
  date_id,
  CUSTOMER_NUMBER,
  STATUS,
  PRODUCT_CODE,
  SALES_REP_ID,
  sum(distinct t.ORDER_NUMBER) as number of orders,
  sum(t.QUANTITY_ORDERED*t.UNIT_PRICE*t.rate_to_eur) as sales amount in EUR,
  max(t.UNIT_PRICE*t.rate_to_eur) as max unit price in EUR,
  avg(t.UNIT_PRICE*t.rate_to_eur) as avg unit price in EUR,
  min(t.UNIT_PRICE*t.rate_to_eur) as min unit price in EUR,
```

sum(t.QUANTITY_ORDERED*t.BUY_PRICE*t.rate_to_eur) as total purchase amount in EUR

from tmp_combined_orders t,

dim_date d,

dim_status s,

dim_product p,

dim_customer c,

dim_sales_rep r,

where t.ORDER_DATE=d.date_id and p.PRODUCT_CODE=t.PRODUCT_CODE and s.STATUS=t.STATUS and c.CUSTOMER_NUMBER=t.CUSTOMER_NUMBER and r.SALES_REP_ID=t.SALES_REP_ID

group by date_id,

PRODUCT_CODE,

STATUS,

CUSTOMER_NUMBER,

SALES_REP_ID;

Q2.10 <8pt.>: Consider the table product_sales created as follows:

```
CREATE TABLE IF NOT EXISTS order_data.products_sales (
ORDER_DATE DATE ,
PRODUCT_CODE STRING,
SALES_AMOUNT DECIMAL(15,2)
)
STORED AS orc;


insert into order_data.products_sales (order_date,product_code,sales_amount)
select to_date(o.order_date) as order_date,
d.product_code,
sum(d.unit_price*d.quantity_ordered) as SALES_AMOUNT
from order_data.stg_order_details d, order_data.stg_orders o
where d.order_number=o.order_number
group by d.product_code,to_date(o.order_date)
```

Provide the statements answering the following questions:

a) For each product_code and order_date, find the SALES_AMOUNT of the current, the previous and the next day.

b) For each product_code and order_date, find the following:
   - the SALES_AMOUNT, and
   - the difference between the SALES_AMOUNT of the product and the SALES_AMOUNT of the product with **maximum** SALES_AMOUNT in the given order_date.

c) For each product_code, find the cumulative sum of SALES_AMOUNT over the order_date.

Hints:

- *Window functions in Hive: link.*
- *In (b), use windows function partitioning by order_date and ordering properly.*
- *In (c), use windows function with corresponding ordering.*

Answer:

Q2.11 <7pt.>: In the following, it is requested to find the association rules (link1, link2) of the following form:

- X => Y; i.e., if X is included in an order, the probability of having Y ordered, as well, is C%.

where X,Y are PRODUCT_CODEs, and C is a given threshold.

Consider also the following:

Support: Supp(**X**)=Number of orders containing all the PRODUCT_CODEs contained in the set **X.**

Confidence: conf(X=>Y)=supp({X} U {Y})/supp({X}).

In particular, provide the statement finding the pairs of products (X,Y), along with their confidence value, for conf(X=>Y) ≤ 80% and supp({X} U {Y}) > 1.

Optional extension (for practice only): *Find the association rules of the form ({X,Y}=>Z), where X, Y, Z are PRODUCT_CODEs.*

Answer: