Big Data Architecture, HDFS and Hive

Piermattia Schoch

PART 1 - Theoretical Question

Q 1.1 Which of the following issues may be caused by lot of small files in HDFS?

A small file is one which is significantly smaller than the HDFS block size (default 64MB). If you're storing small files, then you probably have lots of them (otherwise you wouldn't turn to Hadoop), and the problem is that HDFS can't handle lots of files. More specifically you are facing the following problems:

Namenode memory usage increases significantly:

The NameNode stores the metadata information regarding file system in the RAM. Since each block is considered an object in HDFS taking 150bytes of metadata associated with it, if the the block size is too small, we will have a greater number of blocks and we will might face RAM issue. For example a billions files, each requiring 1 block, the Namenode will require 300GB of memory (reaching the max. capability of current hardware).

Overall network load increase:

Here the problem is due to the fact that in normal operation, the NameNode must constantly track and check where every block of data is stored in the cluster. This is done by listening for data nodes to report on all of their blocks of data. The more blocks a data node must report, the more network bandwidth it will consume.

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

Map tasks process a block of input at a time. When the files are small, it just passes each small file to a map () function, which is not very efficient because it will create a large number of mappers. This 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.

Q 1.2 How is uniform data distribution across the servers achieved in HDFS?

By splitting files into blocks:

The Hadoop Distributed File System (HDFS) distribute storage and computation across many servers. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and the execution of application computations in parallel close to their data. The files are divided into blocks which are distributed on local disks of several cluster nodes. The advantages are that Hadoop cluster scales computation capacity, storage capacity and I/O bandwidth by simply adding commodity servers and that each file can can be larger than any single disk in the network.

Q 1.3 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?

HDFS provides an API that exposes the locations of a file blocks. This allows applications like the MapReduce framework to schedule a task to where the data are located, thus improving the read performance. It also allows an application to set the replication factor of a file. By default a file's replication factor is three. For critical files or files which are accessed very often, having a higher replication factor improves tolerance against faults but increases read bandwidth. (lower performance , greater robustness).
In addition, HDFS should be configured to place block replicas across multiple racks of machines to protect against catastrophic failure of an entire rack or its constituent network infrastructure.

Q. 1.4 You were told that two servers in HDFS were down: Datanode and Namenode. Which would be your reaction?

Restore Namenode first:

NameNode is so critical to HDFS and when the NameNode is down, HDFS/Hadoop cluster is inaccessible and considered down.
When a DataNode is down, it does not affect the availability of data or the cluster. NameNode will arrange for replication for the blocks managed by the DataNode that is not available.

Q 1.5 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? Explain your answer, shortly.

Total number of blocks in HDFS for file:
=> [File size / Block size] * replication factor = (10.000MB / 64MB) * 2 = 312.5

This imply that the file will be splitted among 313 blocks: 312 will be fully occupied, while the last one will have 32MB free.

Then the total disk space required for storing this file is:
=> Number of blocks * Size of blocks = 20.032MB

Q 1.6 Your colleague mistakenly dropped the table 'access_log' created with the following statement, in Hive: Which would be your reaction? Explain your answer, shortly.

For an external table only the metadata of the table is cleared and the data still persists as it is in its place. Hive does not provide a way to retrieve the schema associated, so the only thing to do is to re-write it again. In comparison of dropping an internal table, where you lost the data wherever they reside (regardless any replication), is not a real problem.

Q 1.7 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.

1. Create a directory in HDFS to hold the file.
2. Push the file into this directory.
3. Create an external table in Hive, specifying the location that must be over an existing data storage location (ex: HDFS folder where we pushed the file).
4. The data are now immediately accessible. This data can be queried, but they are not managed by Hive warehouse directory.

1. Create an internal table (ORC or Parquet data-format).
2. Load data directly from local path into the internal table (usually when the format is fixed).

Q 1.8 You have an external table in Hive and want to store this data in more compact and efficient format. Your decision is?

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:

If choose this pipeline to create our tables we are able to store the data in Optimized Row Columnar format (ORC) or in Parquet format. In this way the data are stored in a more efficient and compact (up to 75% lower size) than standard RCFile.

Q 1.9 Why does partitioning optimize Hive queries? Please provide a short answer.

The main reason for which partitioning speed up Hive queries is due to the fact that Table partitioning splits a table into smaller parts that can be accessed, stored, and maintained independent of one another.
A table can have one or more partition columns (partioned vertically) and a separate data directory is created for each distinct value combination in the partition columns. When the table gets queried, only the required partitions (directory) of data in the table are being read, so the I/O and time of the query is greatly reduced

Q 1.10 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.

Hive Map Side Join since one of the tables in the join is a small table and can be loaded into memory. So that a join could be performed within a mapper without using a Map/Reduce step. In this way the Queries are optimized in terms of speed execution.

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

PART 2 - Data Warehousing using Hive

Q 2.1 Provide the statements in order to create the following folder structure within the HDFS folder "cloudera" located in the path /user/cloudera.

```
> hdfs dfs -mkdir /user/cloudera/RawData
> hdfs dfs -mkdir /user/cloudera/RawData/OrdersData
> hdfs dfs -mkdir /user/cloudera/RawData/Rates
```

Q 2.2 Provide the statements in order to load the following data into the corresponding HDFS folder.

```
> hdfs dfs -put /home/cloudera/Desktop/orders_dataset/orders.csv
               /user/cloudera/RawData/OrdersData

> hdfs dfs -put /home/cloudera/Desktop/orders_dataset/order_details.csv
               /user/cloudera/RawData/OrdersData

> hdfs dfs -put /home/cloudera/Desktop/orders_dataset/rates/*.json
              /user/cloudera/RawData/Rates
```

Q 2.3 Provide the statements in order to make the following data accessible through Hive (i.e., the data could be queried through Hive).

```
CREATE DATABASE IF NOT EXISTS order_data;
```

```
USE DATABASE order_data;
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS order_data.ext_orders
(
ORDER_NUMBER BIGINT,
ORDER_DATE string,
SHIPPED_DATE string,
STATUS string,
COMMENTS string,
CUSTOMER_NUMBER string,
CUSTOMER_NAME string,
CUST_CITY string,
CUST_STATE 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 ARRAY<string>,
OFFICE_CITY string,
OFFICE_STATE string ,
OFFICE_TERRITORY string ,
OFFICE_COUNTRY string
)

COMMENT 'This is an external table for orders'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/cloudera/RawData/OrdersData/ext_orders'
tblproperties ("skip.header.line.count "= "1");
```

```
> hdfs dfs -cp /user/cloudera/RawData/OrdersData/orders.csv
              /user/cloudera/RawData/OrdersData/ext_orders
```

"For this particular table I had a problem in removing the headers using the above command "tblproperties …" then I decided to remove it manually"

```
CREATE EXTERNAL TABLE IF NOT EXISTS order_data.ext_order_details
(
ORDER_NUMBER BIGINT,
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
)

COMMENT 'This is an external table for order_details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION '/user/cloudera/RawData/OrdersData/ext_order_details'
tblproperties ("skip.header.line.count" = "1");
```

```
> hdfs dfs -cp /user/cloudera/RawData/OrdersData/order_details.csv
             /user/cloudera/RawData/OrdersData/ext_order_details
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS order_data.ext_rates
(
rates string
)

STORED AS TEXTFILE
LOCATION '/user/cloudera/RawData/Rates/ext_rates'
```

```
> hdfs dfs -cp /user/cloudera/RawData/Rates/*.json
               /user/cloudera/ RawData/Rates/ext_rates
```

Q 2.4 Provide the statements in order to create a partitioned table named "stg_orders" for storing historical orders data. Select proper partition columns.

```
CREATE VIEW order_data.view_orders6
as
SELECT

order_number,

cast(to_date(from_unixtime(unix_timestamp(order_date,'yyyy-MM-
dd'),'yyyy-MM-dd')) as date) as order_date,

month(cast(to_date(from_unixtime(unix_timestamp(order_date,'yyyy-MM-
dd'),'yyyy-MM-dd')) as date)) as order_month,

year(cast(to_date(from_unixtime(unix_timestamp(order_date,'yyyy-MM-
dd'),'yyyy-MM-dd')) as date)) as order_year,

cast(to_date(from_unixtime(unix_timestamp(shipped_date,'yyyy-MM-
dd'),'yyyy-MM-dd')) as date) as shipped_date,

status,
comments,
cast(customer_number as int) as customer_number,
customer_name,
cust_city,
cust_state,
cust_country_iso,
sales_currency,
cast(sales_rep_id as smallint) as sales_rep_id,
sales_rep_firstname,
sales_rep_lastname,
cast(office_code as int) as office_code,
reporting_path,
office_city,
office_state,
office_territory,
office_country
FROM order_data.ext_orders
```

The casting operations for date columns seems to work properly. Running cast(column as date) returns NULL values due to the format in which the value are stored in the csv file.

view_orders6.order_date	view_orders6.order_month	view_orders6.order_year	view_orders6.shipped_date	view_orders
2004-07-20	7	2004	2004-07-23	Shipped
2004-07-07	7	2004	2004-07-09	Shipped
2004-05-11	5	2004	2004-05-15	Shipped
2004-04-02	4	2004	2004-04-06	Shipped
2004-02-22	2	2004	2004-02-24	Shipped

```
CREATE  TABLE IF NOT EXISTS order_data.stg_orders
(
ORDER_NUMBER BIGINT,
ORDER_DATE date,
SHIPPED_DATE date,
STATUS string,
COMMENTS string,
CUSTOMER_NUMBER int,
CUSTOMER_NAME string,
CUST_CITY string,
CUST_STATE string,
CUST_COUNTRY_ISO string,
SALES_CURRENCY string,
SALES_REP_ID smallint,
SALES_REP_FIRSTNAME string,
SALES_REP_LASTNAME string,
OFFICE_CODE int,
REPORTING_PATH ARRAY<string>,
OFFICE_CITY string,
OFFICE_STATE string ,
OFFICE_TERRITORY string ,
OFFICE_COUNTRY string
)
PARTITIONED BY (ORDER_MONTH int, ORDER_YEAR int)
STORED AS ORC;
```

Q 2.5 Provide the statements loading the data from the external tables to the stg_tables by converting the columns into the proper data type.

```
create view order_data.view_order_details
as
SELECT

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_ordered,
cast(unit_price as DECIMAL(15,2)) as unit_price

FROM order_data.ext_order_details
```

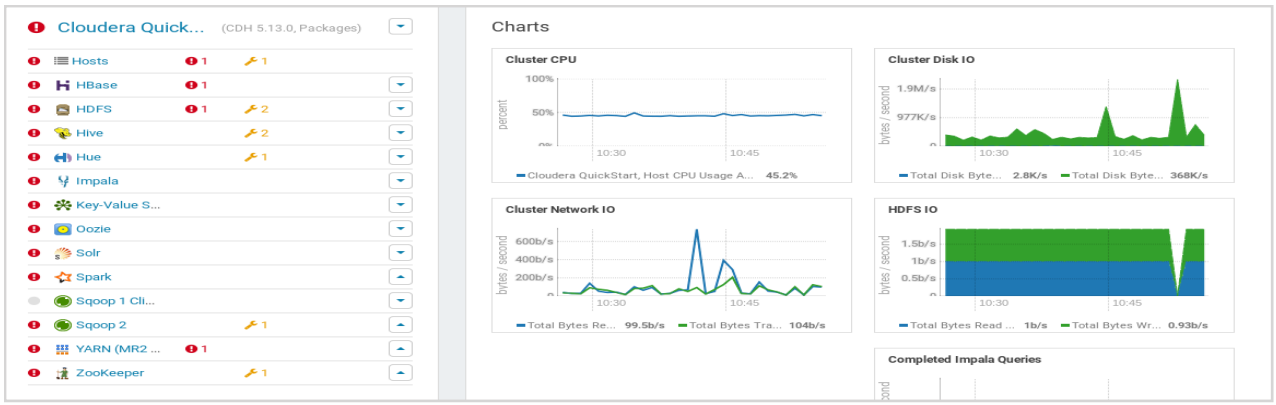
```
CREATE TABLE IF NOT EXISTS order_data.stg_order_details (
ORDER_NUMBER BIGINT,
PRODUCT_CODE STRING,
PRODUCT_NAME STRING,
PRODUCT_CATEGORY STRING,
PRODUCT_VENDOR STRING,
QUANTITY_IN_STOCK INT,
BUY_PRICE DECIMAL(15,2),
QUANTITY_ORDERED INT,
UNIT_PRICE DECIMAL(15,2)
)
STORED AS orc;
```

```
CREATE VIEW IF NOT EXISTS order_data.view_rates
  AS
  SELECT

cast(get_json_object(rates,'$.date') as date) as indate,
get_json_object(rates,'$.rates') as rates

FROM order_data.ext_rates;
```

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

Now it's time to insert data into the staging table. Unfortunately I wasn't able to load properly the data into the stg_orders table. Since the assignment can be executed without coding, I will provide the chunks of code where I got stuck with their relative error, then for the following questions (Q6 —> ..) I continued using pen and paper. I constantly had health issue on Cloudera manager.

1) I constantly had health issue on Cloudera manager (restarted properly then bad health)



2) Loading the data into the stg_order using Dynamic Partionioning gave some problem due to the casting made in Q3

```
INSERT INTO order_data.stg_orders
PARTITION(ORDER_YEAR, ORDER_MONTH)

SELECT
order_number,
order_date,
order_month,
order_year,
shipped_date,
status,
comments,
customer_number,
customer_name,
cust_city,
cust_state,
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 order_data.view_orders6;
```

```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

```
hive> INSERT INTO order_data.stg_orders
    > PARTITION(ORDER_YEAR, order_month)
    >
    > SELECT
    > order_number,
    > order_date,
    > order_month,
    > order_year,
    > shipped_date,
    > status,
    > comments,
    > customer_number,
    > customer_name,
    > cust_city,
    > cust_state,
    > 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 order_data.view_orders6;
FAILED: UDFArgumentException CAST as DATE only allows date,string, or timestamp
types
hive>
```

Then I decided to not go ahead with partitioning and I created a new view table (without extracting month and year) and a new staging table not partioned. In this way I had a more serious error

```
Query ID = cloudera_20190602110707_f0504638-4ac9-42ed-84fa-15c18d5e3668
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1559492419992_0005, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1559492419992_
0005/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1559492419992_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2019-06-02 11:07:26,425 Stage-1 map = 0%,  reduce = 0%
2019-06-02 11:07:56,245 Stage-1 map = 100%,  reduce = 0%
Ended Job = job_1559492419992_0005 with errors
Error during job, obtaining debugging information...
Examining task ID: task_1559492419992_0005_m_000000 (and more) from job job_1559492419992_0005

Task with the most failures(4):
-----
Task ID:
  task_1559492419992_0005_m_000000

URL:
  http://quickstart.cloudera:8088/taskdetails.jsp?jobid=job_1559492419992_0005&tipid=task_1559492419992_0005_m_000000
-----
Diagnostic Messages for this Task:
Error: Java heap space

FAILED: Execution Error, return code 2 from org.apache.hadoop.hive.ql.exec.mr.MapRedTask
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   HDFS Read: 0 HDFS Write: 0 FAIL
Total MapReduce CPU Time Spent: 0 msec
hive>
```

```
INSERT INTO order_data.stg_order_details
(
order_number,
product_code,
product_name,
product_category,
product_vendor,
quantity_in_stock,
buy_price,
quantity_ordered,
unit_price
)
SELECT
order_number,
product_code,
product_name,
product_category,
product_vendor,
quantity_in_stock,
buy_price,
quantity_ordered,
unit_price
FROM order_data.view_order_details;
```

order_data.stg_order_details

Columns | Details | Sample

	stg_order_details.order_number	stg_order_details.product_code	stg_order_details.product_name	stg_order_detail
31	10104	S18_3232	1992 Ferrari 360 Spider red	Classic Cars
32	10104	S18_4027	1970 Triumph Spitfire	Classic Cars
33	10104	S24_1444	1970 Dodge Coronet	Classic Cars
34	10104	S24_2840	1958 Chevy Corvette Limited Edition	Classic Cars
35	10104	S24_4048	1992 Porsche Cayenne Turbo Silver	Classic Cars
36	10104	S32_2509	1954 Greyhound Scenicruiser	Trucks and Buses
37	10104	S32_3207	1950's Chicago Surface Lines Streetcar	Trains
38	10104	S50_1392	Diamond T620 Semi-Skirted Tanker	Trucks and Buses

```
INSERT INTO order_data.stg_conv_rate
(indate,
rates)
SELECT indate,
rates
FROM order_data.view_rates;
```

order_data.stg_conv_rate

Columns | Details | Sample

	stg_conv_rate.indate	stg_conv_rate.rates
31	2003-01-31	{"BGN":1.9559,"NZD":1.8668,"TRL":1893200,"CAD":1.6372,"USD":1.3317,"CHF":1.5368,"SKK":38.785,"ZAR":7.6426,"AUD":
32	2003-02-01	{"BGN":1.9559,"NZD":1.8877,"CAD":1.6278,"USD":1.3507,"CHF":1.5444,"SKK":38.655,"ZAR":7.5893,"AUD":1.7329,"JPY":1
33	2003-02-02	{"BGN":1.9558,"NZD":1.8274,"TRL":1852700,"CAD":1.5603,"USD":1.2304,"CHF":1.5532,"SKK":40.015,"ZAR":8.0533,"AUD":

Q 2.6 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 IF NOT EXISTS order_data.dim_keys
(
  dim_key int,
  dim_type string,
  value string,
  created_at timestamp
)
STORED AS orc;
```

```
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;
```

```
CREATE TABLE IF NOT EXISTS order_data.dim_status(
    STATUS_KEY INT,
    STATUS STRING
)
STORED AS orc;
```

```
INSERT INTO order_data.dim_status
(STATUS_KEY, STATUS)
SELECT
/*+ MAPJOIN(k) */
k.dim_key ,s.STATUS_KEY, s.STATUS
FROM (select k.value ,k.dim_key from order_data.dim_keys k where
k.dim_type = 'STATUS') k
join(select s.STATUS
        from order_data.stg_orders s
        group.by s.STATUS) s
on k.value = s.STATUS;
```

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

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

```
INSERT INTO order_data.dim_product
(PRODUCT_KEY, PRODUCT_CODE, PRODUCT_NAME,
PRODUCT_CATEGORY,PRODUCT_VENDOR)
SELECT
/*+ MAPJOIN(k) */
k.dim_key, p.PRODUCT_CODE, p.PRODUCT_NAME, p.PRODUCT_CATEGORY,
p.PRODUCT_VENDOR
FROM order_data.dim_keys k
join(select p.PRODUCT_CODE, p.PRODUCT_NAME, p.PRODUCT_CATEGORY,
p.PRODUCT_VENDOR
from order_data.stg.order_details p
group by p.PRODUCT_CODE, p.PRODUCT_NAME, p.PRODUCT_CATEGORY,
p.PRODUCT_VENDOR) p
on k.dim_type = 'PRODUCT_CODE' and k.VALUE = p.PRODUCT_CODE;
```

Q. 7 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;
```

```
INSERT INTO order_data.dim_customer
(CUST_KEY,
 CAST(CUSTOMER_NUMBER as string) AS CUSTOMER_NUMBER,
 CUSTOMER_NAME,
 CUST_CITY,
 CUST_STATE,
 CUST_COUNTRY,
 CUST_COUNTRY_ISO)
SELECT k.dim_key,
c.CUSTOMER_NUMBER,
c.CUSTOMER_NAME,
c.CUST_CITY,
c.CUST_STATE,
c.CUST_COUNTRY,
c.CUST_COUNTRY_ISO
FROM order_data.dim_key k
JOIN (SELECT c.CUSTOMER_NUMBER,
c.CUSTOMER_NAME,
c.CUST_CITY,
c.CUST_STATE,
c.CUST_COUNTRY,
c.CUST_COUNTRY_ISO
        FROM order_data.stg_order_details
        GROUP BY c.CUSTOMER_NUMBER, c.CUSTOMER_NAME, c.CUST_CITY,
c.CUST_STATE, c.CUST_COUNTRY, c.CUST_COUNTRY_ISO) c
ON k.dim_type ='customer' and k.value=p.CUSTOMER_NUMBER;
```

```
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;
```

```
INSERT INTO order_data.dim_sales_rep
(
SALES_REP_KEY,
CAST(SALES_REP_ID INT AS STRING) AS SALES_REP_ID,
SALES_REP_FIRSTNAME,
SALES_REP_LASTNAME,
OFFICE_CODE,
OFFICE_CITY,
OFFICE_STATE,
OFFICE_TERRITORY,
OFFICE_COUNTRY
)

SELECT k.dim_key,
s.SALES_REP_ID, s.SALES_REP_FIRSTNAME, s.SALES_REP_LASTNAME,
s.OFFICE_CODE, s.OFFICE_CITY, s.OFFICE_STATE, s.OFFICE_TERRITORY,
s.OFFICE_COUNTRY
FROM order_data.dim_keys k
JOIN (SELECT s.SALES_REP_ID, s.SALES_REP_FIRSTNAME,
s.SALES_REP_LASTNAME, s.OFFICE_CODE, s.OFFICE_CITY, s.OFFICE_STATE,
s.OFFICE_TERRITORY, s.OFFICE_COUNTRY
        FROM order_data.stg_orders s
        GROUP BY s.SALES_REP_ID, s.SALES_REP_FIRSTNAME,
s.SALES_REP_LASTNAME, s.OFFICE_CODE, s.OFFICE_CITY, s.OFFICE_STATE,
s.OFFICE_TERRITORY, s.OFFICE_COUNTRY) s
ON k.dim_type='salesrep' and k.value=p.SALES_REP_ID;
```

Q. 8 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;
```

```
with min_max dates as (
select min(ORDER_DATE) AS MIN_ORDER_DATE, current_date() AS TODAY
from order_data.stg_orders
)
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,
year(date) as year,
month(date) ad 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;
```