SCALA
See: Orange - Spark Scala - Part 1.html

Which contains the tasks completed in Databricks community edition.

GIT
1. Explain, how to resolve a conflict in Git?

In Git, it is possible that during the act of integrating a current working branch (e.g. feature_1) with another branch (e.g. develop), the software cannot know what changes are correct.

For instance, it is possible that two people changed the same lines in the same file or one person decided to delete a file and the other decided to modify it. In these cases, a conflict is raised.

In order to solve it:

- It is necessary to know what has happened, and why more than one person has modified the same lines of a certain file (or deleted it).
- Have a look at the conflicted file/s:
  - Git encloses the problematic area with: "<<<<<<< HEAD" and ">>>>>>> [other/branch/name]", telling us from which branch the conflict occurs.
- Clean up the lines with the previous marks and leave the file as expected using a text editor or within the IDE (Integrated Development Environment).
- Mark the file as resolved by hand with “git add”.
- Conclude with a regular “git commit”.

It is possible to use a CLI tool (e.g. git bash in Windows) to execute the commands or even an integrated tool in an IDE (e.g. IntelliJ IDEA) through a graphical interface.

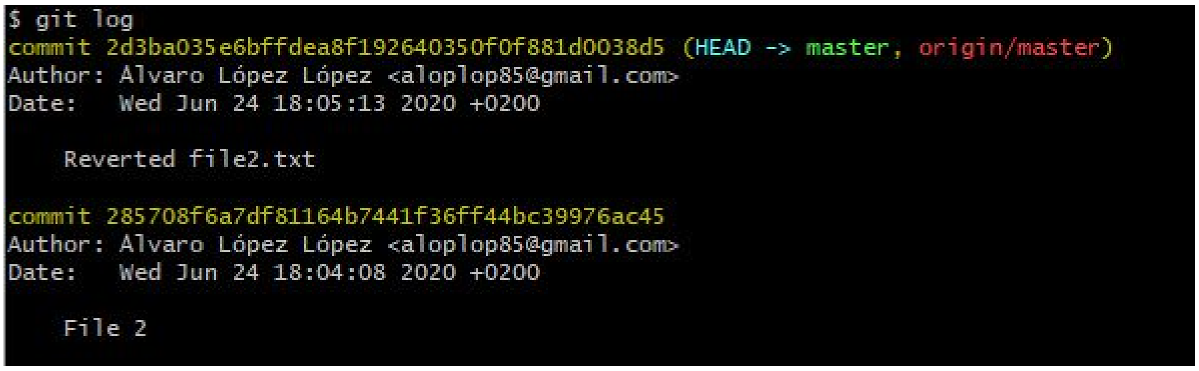
2. How do you revert a commit that has already been pushed and made public?

Revert means to undo the changes introduced by the commit. The general rule in this case it is that you should not rewrite the published history, because somebody might have based their work in it.

The main solution would be to create a new commit which reverts the changes pushed. One solution is based in **git-revert**:

```
git revert  --no-commit <commit>
git commit -m “Last commit reverted”
git push -u origin <branch>
```

The next figure is a capture of the “git log” output where a commit of a file called “file2.txt” was done to GitHub. Using git revert the upload was undone.



```
$ git log
commit 2d3ba035e6bffdea8f192640350f0f881d0038d5 (HEAD -> master, origin/master)
Author: Alvaro López López <aloplop85@gmail.com>
Date:   Wed Jun 24 18:05:13 2020 +0200

    Reverted file2.txt

commit 285708f6a7df81164b7441f36ff44bc39976ac45
Author: Alvaro López López <aloplop85@gmail.com>
Date:   Wed Jun 24 18:04:08 2020 +0200

    File 2
```

There exist alternatives with reset and push with a “force” flag, but it is not desirable to execute them, as well as just the Git administrator should take care of them:

```
git reset --hard <commit>
git push -f origin <branch>
```

3. Explain the difference between git pull and git fetch?

git pull: it fetches the commits done in a repository and **integrates** them with another repository or to a local branch. E.g. git pull origin <branch>. In fact, it is a combination of git fetch and git merge.

git fetch: brings branches, tags and objects from one or more repositories to the local history to complete it, but **it doesn’t integrate** the changes done to these branches into the local copy of them.

An example scenario could be as follows: three people are working on the “develop” branch (P1, P2, P3):

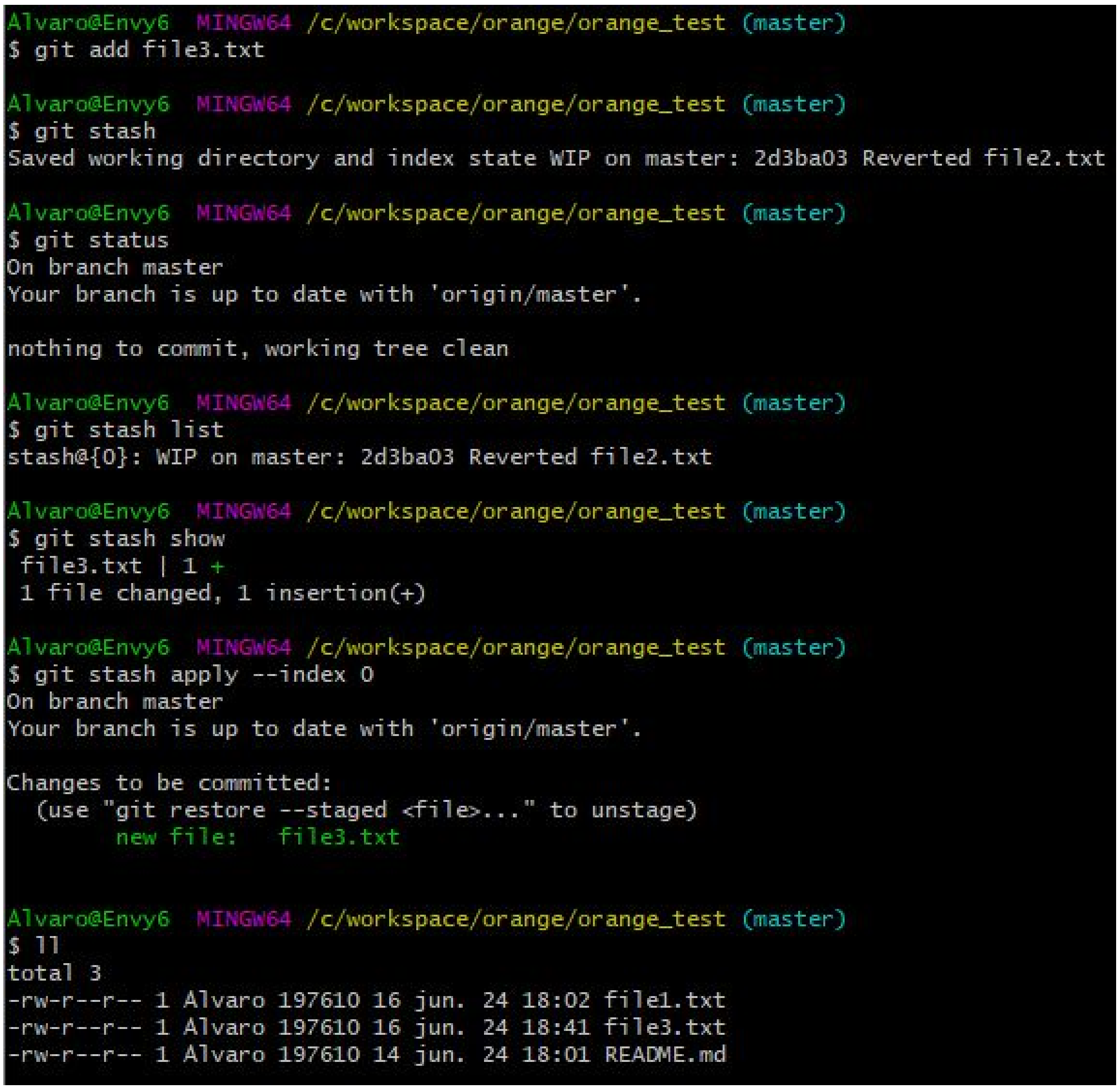
- P2 completes a development and pushes the changes to the “develop” branch.
- If P1 executes “git fetch”, she just gets the commit done by P2, but changes are not integrated into her local copy of the branch.
- If P3 executes “git pull”, she would get the commit done and would integrate the changes into the local copy, so she could detect any conflict with her development.

4. What is git stash?

This command is used when a developer wants to go back to a clean directory (i.e. revert the state of the project to match the HEAD commit) without losing the current state of the work. These changes are created into the “stash”, a stack of unfinished changes which can be applied whenever they are required.

It is possible to call this command several times, and list them using “`git stash list`”, inspect them with “`git stash show`”, and restore them with “`git stash apply`”.

In the next picture, file3.txt is added to the index, a "`git stash call`" is made so the file is pushed into the stash stack. Finally, using "`git stash apply --index 0`" the file is reverted back into the folder.



```
Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ git add file3.txt

Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ git stash
Saved working directory and index state WIP on master: 2d3ba03 Reverted file2.txt

Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ git stash list
stash@{0}: WIP on master: 2d3ba03 Reverted file2.txt

Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ git stash show
 file3.txt | 1 +
 1 file changed, 1 insertion(+)

Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ git stash apply --index 0
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file3.txt


Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ ll
total 3
-rw-r--r-- 1 Alvaro 197610 16 jun.  24 18:02 file1.txt
-rw-r--r-- 1 Alvaro 197610 16 jun.  24 18:41 file3.txt
-rw-r--r-- 1 Alvaro 197610 14 jun.  24 18:01 README.md
```

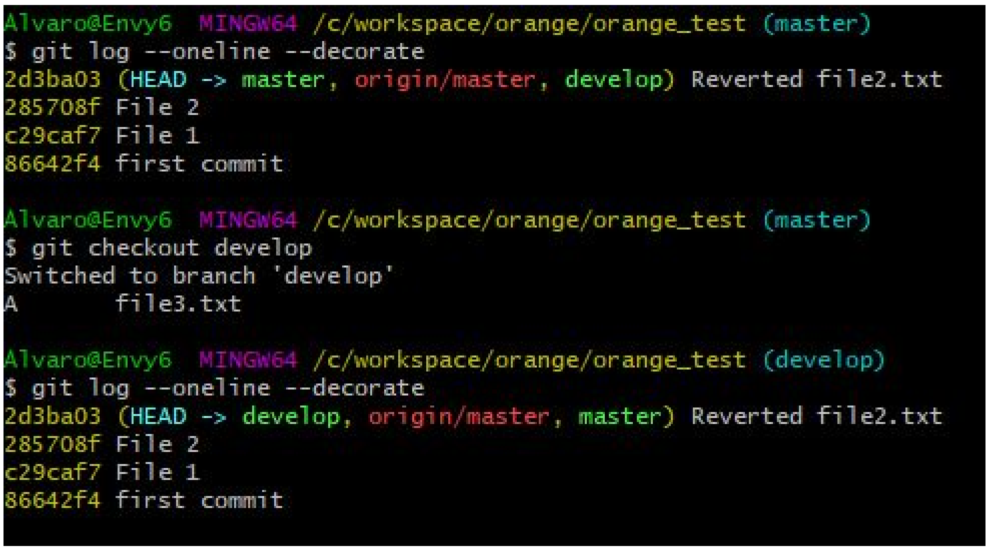
5. What is git fork? What is the difference between fork, branch and clone?

Fork in Git means to create a copy of a project repository source code to another Git repository under the user control (e.g. a GitHub profile repository). This operation is done to propose changes to the source project or develop a new idea using the original project as a starting point.

Git **clone** is used to copy a repository into a newly created directory, including remote-tracking branches for each branch in the cloned repository, visible with "`git branch --remotes`". Whenever you create a repository (e.g. in GitHub), in order to work in this repo locally, you must execute a git clone command.

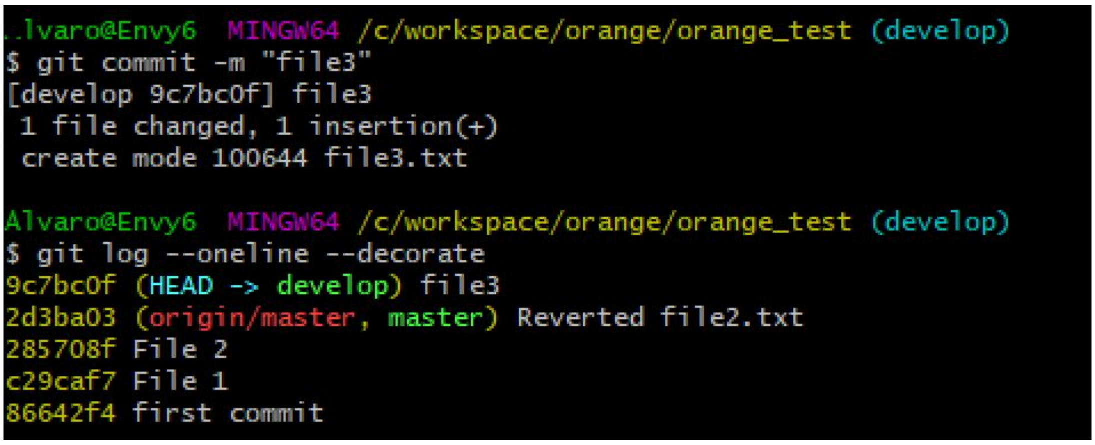
A git **branch** is a pointer to the commits done to the source code version control. Creating a new branch implies that a new pointer is created pointing to the same commit the user is working on. This way, the new pointer can move forward, independently of the first one -i.e. a new development can be done while keeping the other branch in the same state-..

In the next Figure, it is possible to see that after creating a new branch called “develop”, the pointer is in the same position (2d3ba03):



```
Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ git log --oneline --decorate
2d3ba03 (HEAD -> master, origin/master, develop) Reverted file2.txt
285708f File 2
c29caf7 File 1
86642f4 first commit

Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (master)
$ git checkout develop
Switched to branch 'develop'
A       file3.txt

Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (develop)
$ git log --oneline --decorate
2d3ba03 (HEAD -> develop, origin/master, master) Reverted file2.txt
285708f File 2
c29caf7 File 1
86642f4 first commit
```

After a commit is done in the “develop” branch, the “master” and “develop” branches’ pointers are detached, as in the next figure:



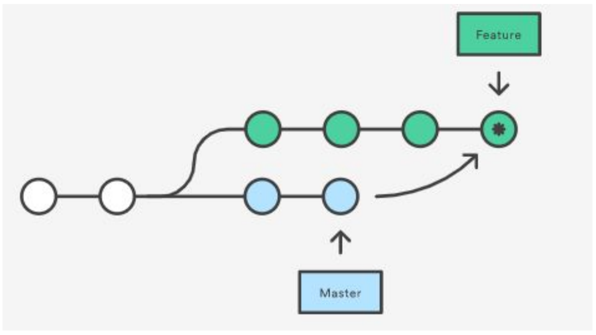
```
.lvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (develop)
$ git commit -m "file3"
[develop 9c7bc0f] file3
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt

Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (develop)
$ git log --oneline --decorate
9c7bc0f (HEAD -> develop) file3
2d3ba03 (origin/master, master) Reverted file2.txt
285708f File 2
c29caf7 File 1
86642f4 first commit
```

6. What is the difference between rebasing and merge in Git?

Both commands integrate changes from one branch into another, however, they do it in different ways:

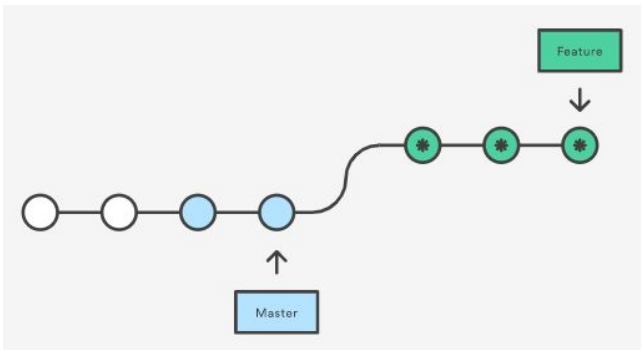
- **git merge** incorporates the commits done on another branch (i.e. relevant commits) by creating a "merge commit", which ties the development history of both branches. Furthermore, it is a non-destructive operation. The following picture shows how the commits in the master branch are being tied with the feature branch with the merge commit marked with an asterisk:

  ```
  git checkout feature
  git merge master
  ```



- **git rebase** moves the current branch on the top of the other one where the relevant commits were made. It also re-writes the project history by creating new commits for each commit in the original branch. The major benefit is obtaining a cleaner project history. In the following picture, the feature branch has been moved on the top of master, with additional commits marked with an asterisk.

  ```
  git checkout feature
  git rebase master
  ```

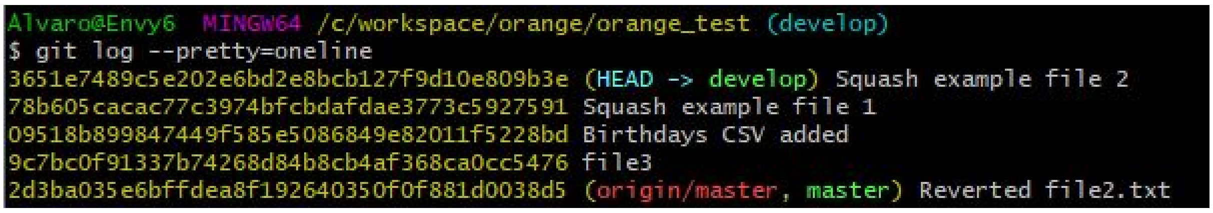


7. How do you squash the last N commits into a single commit?

Squash the last N commits is a good way to group changes before sharing with other colleagues by making the development history cleaner. It is done by using the git rebase command:

git rebase -i HEAD~N
Where N is the number of the last N commits

Example of the last three commits in the “develop” branch:



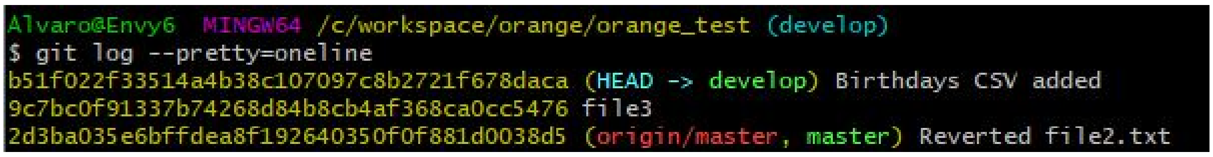
```
Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (develop)
$ git log --pretty=oneline
3651e7489c5e202e6bd2e8bcb127f9d10e809b3e (HEAD -> develop) Squash example file 2
78b605cacac77c3974bfcbdafdae3773c5927591 Squash example file 1
09518b899847449f585e5086849e82011f5228bd Birthdays CSV added
9c7bc0f91337b74268d84b8cb4af368ca0cc5476 file3
2d3ba035e6bffdea8f192640350f0f881d0038d5 (origin/master, master) Reverted file2.txt
```

Select to squash the last three commits: `git rebase -i HEAD~3`
pick 09518b8 Birthdays CSV added
squash 78b605c Squash example file 1
squash 3651e74 Squash example file 2

Another text editor appears:
# This is a combination of 3 commits.
# This is the 1st commit message:
Birthdays CSV added
# This is the commit message #2:
Squash example file 1
# This is the commit message #3:
Squash example file 2

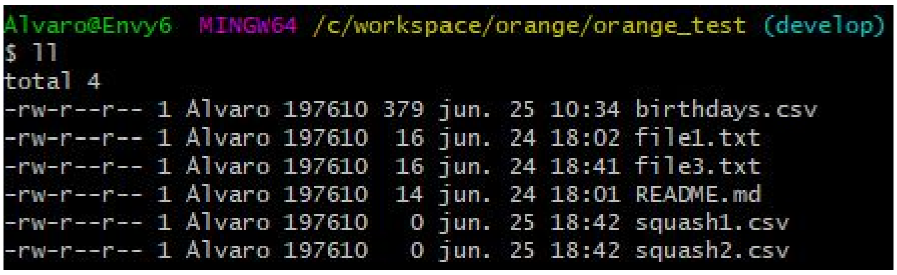
$ git rebase -i HEAD~3
[detached HEAD b51f022] Birthdays CSV added
 Date: Thu Jun 25 18:39:16 2020 +0200
 3 files changed, 26 insertions(+)
 create mode 100644 **birthdays.csv**
 create mode 100644 **squash1.csv**
 create mode 100644 **squash2.csv**
Successfully rebased and updated refs/heads/develop.

Just the last commit remains:



```
Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (develop)
$ git log --pretty=oneline
b51f022f33514a4b38c107097c8b2721f678daca (HEAD -> develop) Birthdays CSV added
9c7bc0f91337b74268d84b8cb4af368ca0cc5476 file3
2d3ba035e6bffdea8f192640350f0f881d0038d5 (origin/master, master) Reverted file2.txt
```

Final check to verify the three files are currently in the folder:



```
Alvaro@Envy6  MINGW64 /c/workspace/orange/orange_test (develop)
$ ll
total 4
-rw-r--r-- 1 Alvaro 197610 379 jun. 25 10:34 birthdays.csv
-rw-r--r-- 1 Alvaro 197610  16 jun. 24 18:02 file1.txt
-rw-r--r-- 1 Alvaro 197610  16 jun. 24 18:41 file3.txt
-rw-r--r-- 1 Alvaro 197610  14 jun. 24 18:01 README.md
-rw-r--r-- 1 Alvaro 197610   0 jun. 25 18:42 squash1.csv
-rw-r--r-- 1 Alvaro 197610   0 jun. 25 18:42 squash2.csv
```

HIVE

1. What is the difference between external table and managed table?

The fundamental difference is that Hive assumes that **it owns the data for managed tables**. This means that data, properties and data layout can only be changed via a Hive command. Furthermore, data is attached to a Hive entity so, in case the user changes an entity (e.g. drop a table), the data is also changed (e.g: if a table is dropped, the data is deleted).

An external table points to any location supported for its storage. In order to create a external table, the EXTERNAL keyword must be used:

(Managed table)
CREATE TABLE IF NOT EXISTS employee (id INT, name STRING)
COMMENT ‘Managed table for employees’
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE;

(External table)

CREATE **EXTERNAL** TABLE IF NOT EXISTS employee_ext (id INT, name STRING)
COMMENT ‘Managed table for employees’
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE
LOCATION ‘/employee_data’;

2. Is it possible to change the default location of a managed table?

Yes, by using the LOCATION keyword along with the path while creating it -by default it is /user/hive/warehouse-.
Furthermore it is possible to use the ALTER TABLE command after creating it:

```
ALTER TABLE employee SET LOCATION ‘/managed_data/employee’;
```

3. When should we use SORT BY instead of ORDER BY?

We can use **SORT BY** if the total order of the query is not relevant, or the aggregated results can be reordered afterwards, with a smaller set of records, as this command only guarantees the **ordering of the rows within a reducer** (assuming hive.execution.engine=mr). Therefore, the result consists of N sorted files with overlapping ranges.

ORDER BY is used to **guarantee the total order** in the output by pushing all data through just one reducer (i.e. one sorted file as output), which can translate into unacceptable times for large datasets.

4. Why do we perform partitioning in Hive?

Partitioning is a way of dividing a table based on the values of particular columns. This way it is easy to query slices of the data, not the full dataset (e.g. a range of dates). Therefore, the I/O time required by the query decreases.

In order to create a partitioned table, the PARTITIONED BY option is used:

```
-- A partitioned table by office for employees
CREATE TABLE IF NOT EXISTS employee_office ( id INT, name STRING)
PARTITIONED BY (office STRING)
COMMENT ‘Managed table for employees partitioned by office’
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE;
```

If we have the employees in separated files per office, we could insert into the table as follows:

```
LOAD DATA LOCAL INPATH ‘/office/madrid’ OVERWRITE INTO TABLE employee_office PARTITION (office = ‘Madrid’)
LOAD DATA LOCAL INPATH ‘/office/barcelona’ OVERWRITE INTO TABLE employee_office PARTITION (office = ‘Barcelona’)
```

Then, if our queries are mainly used by office, the performance will be better than if the table were not partitioned:

```
SELECT id, name
FROM employee
WHERE office = ‘Madrid’ OR office = ‘Barcelona’
```

5. What is dynamic partitioning and when is it used?

Dynamic partitioning is related to partitioning columns in a Hive table but the values or these columns are only known at **execution time**. Usually it is used when:

- The user has large data stored in a non-partitioned table and wants to work in a "partitioned" way.
- The user wants to partition a number of columns but this number is not known.
- The user does not want to perform an alter operation afterwards.

To support dynamic partitioning, the hive.mapred.mode must be set to **nonstrict** and hive.exec.dynamic.partition option must be set to true.

On the other hand, in static partitioning the values are known at compile time.

An example of moving data with a dynamic partitioning scheme would be as follows:

```
INSERT OVERWRITE TABLE employee_office PARTITION (office)
SELECT id, name, office
FROM employee
-- WHERE conditions...
```

In case of static partitioning (note switching of 'madrid' to 'Madrid':

```
INSERT OVERWRITE TABLE employee_office PARTITION (office = 'Madrid')
SELECT id, name /*, office */
FROM employee
WHERE office IS NOT NULL
  AND office = 'madrid'
```

6. How does data transfer happen from HDFS to Hive?

There doesn't exist a data transfer between HDFS and Hive. Hive is just a metadata layer which helps the user to access the HDFS data using a traditional database and tables structure with a subset of operations of the SQL standard, known as HQL (Hive Query Language).

KAFKA
1. Describe architecture Kafka

Kafka is a publish/subscribe messaging system with the following components:

- **Messages and batches**: the unit of data within Kafka is a message (similar to a row or a record in a table). It is simply an array of bytes without a specific meaning to Kafka. A message can optionally contain metadata, which is known as the key (i.e: key-value pairs). For efficiency, messages are written into batches (i.e: a collection of messages) which are produced to the same topic and partition.
- **Schemas:** it is recommended that additional structure, or scheme, be imposed to the message content in order to be understood. Examples are: CSV, JSON, XML or serialized with Avro -a serialization framework developed for Hadoop-.
- **Topics and Partitions**: messages in Kafka are categorized into topics (e.g. ‘clients’, ‘notifications’). Topics are broken down into a certain number of partitions, which are single logs. Messages are written in an append-only fashion and read in order from beginning to end. There is no message time-ordering across the entire topic, just within a single partition.
- **Producers and Consumers:** producers create new messages and publish them to a specific topic. By default, producers do not care about what partition a specific message is written to, but in certain cases they direct messages to specific partitions through the partition key (see point 3). Consumers read messages. They subscribe to one or more topics and read the messages in the order in which they were produced. Each consumer keeps track of which messages has already consumed by keeping track of the offset messages -the offset is an increasing integer value-. Consumers work as part of a **consumer group**, in which one or more consumers work together to consume a topic. The consumer group assures each partition is only consumed by one member. The mapping of a consumer to a partition is often called **ownership**.
- **Brokers and Clusters:** a single Kafka server is called a broker. I receives messages from producers, assigns offsets to them and commit the messages to storage on the disk. Kafka brokers are designed to operate as part of a cluster, where one of them will also function as the cluster **controller**. It is responsible for administrative operations, including assigning partitions to brokers and monitoring for broker failures. A partition is owned by a single broker in the cluster and that broker is called the **leader** of the partition. A partition may be assigned to multiple brokers, which will result in the partition being replicated, providing redundancy of the messages in a partition. All consumers and producers operating on a partition must connect to the leader.
- **Zookeeper:** it is used to manage the cluster (detailed answer in next section).

2. What role does ZooKeeper play in a cluster of Kafka?

Kafka uses Zookeeper to store metadata about the Kafka cluster and consumer client details, such as:

- Offset of the last consumed message for each partition (Zookeeper or Kafka itself).
    - It is recommended to use the latest Kafka libraries, committing offsets directly to Kafka, removing the dependency on Zookeeper.
- Kafka brokers metadata
    - Topics
    - Partitions
- List of brokers that are currently members of a cluster (a heartbeat is sent to Zookeeper to control this).
- Consumer groups -old libraries, newer are maintained in Kafka brokers-.

Zookeeper usually runs in a cluster, which is called “ensemble” and provides a flexible and robust synchronization in the Kafka cluster. Each time a broker process starts, it registers itself with its ID in Zookeeper by creating an ephemeral node. Kafka uses this feature to elect a **controller** and to notify the controller when nodes join and leave the cluster. A controller is one of the brokers which, in addition to the normal functionality, is responsible for electing partition leaders.

3. Describe Partitioning key

Kafka messages are key-value pairs so applications may produce records with a null key or with some value. The key has two goals:

- Include additional information that gets stored with the message.
- **Decide which one of the topic partitions the message will be written to.** Therefore, all messages with the same key will be written to the same partition. If a process is reading only a subset of the partitions in a topic, all the records for a single key will be read by the same process.

In case there exists a null value for the key, the default partitioner is used, so the record will be sent to one of the available partitions of the corresponding topic in a random way.

If a key exists and the default partitioner is used, Kafka hashes the key and use the result to map the message to a specific partition. Therefore, one key is always mapped to the same partition.

The mapping of keys to partitions is consistent only as long as the number of partitions in a topic does no change.

It is also possible to implement a custom partitioning strategy. E.g. A certain repeated value in a high rate is given one partition and the rest of the keys are hash partitioned.

4. Describe replication system of Kafka

The replication system is a critical issue in Kafka because it guarantees availability and durability when an individual node fails.

Each Kafka topic is partitioned, and each partition can have multiple replicas, which are stored on brokers -each broker stores replicas belonging to different topics and partitions-. There exists two type of replicas:

- **Leader replica**: each partition has a single replica designated as the leader. All produce and consume requests go through the leader to guarantee consistency.
  The leader also knows which follower replicas are up-to-date.
- **Follower replica**: their job is to replicate messages from the leader and stay up-to-date with the most recent messages the leader has. If a leader replica crashes, one of the follower replicas will be promoted to become the new leader of the partition.
  In other to stay in sync with the leader, the replicas send Fetch requests, which contains the offset of the message that the replica wants to receive next.
  Replicas that are constantly asking for the latest messages are called **in-sync replicas**, which are the only ones eligible to be elected as partition leader. On the other hand, replicas that hasn't caught up with the leader in 10 seconds are called **out of sync replicas**.

In addition to the current leader, each partition has a **preferred leader**, which is the replica that was the leader when the topic was originally created. Initially, when partitions are first created, the leaders are **balanced** between brokers. By default, Kafka checks if the preferred leader replica is not the current leader in order to trigger a leader election to make the preferred leader as the current leader if the replica is in-sync.

Part 2 – Practical Use Case
Create a project (using scala, hdfs, hive…) with the following parts. It's recommendable using a good structure for all project (conf files, hql, gradle or sbt…). The project must be contain the following steps:

***** The full project is published in GitHub: https://github.com/lopezavila85/orange_test**

Section 1 is included in the project resources. Tests completed in CDH 5.13 Cloudera quickstart virtual machine. The full section one is included in "hdfs_hive_birthday.sh", a shell script which invokes the HDFS and Hive query commands required.

For instance, this script would be executed during the deployment of the project in order to load required data in HDFS and download certain results from Hive.

Section 2 has been programmed with Scala classes and tests.

1. Hive and Hdfs.
1.1. Load the csv attachment in hdfs.

A CSV with two columns (id, birthday) have been created for this use case.

```
# Create HDFS folder
hdfs dfs -mkdir -p /user/cloudera/orange/birthday

# Load birthdays CSV to HDFS
hdfs dfs -put birthdays.csv /user/cloudera/orange/birthday
```

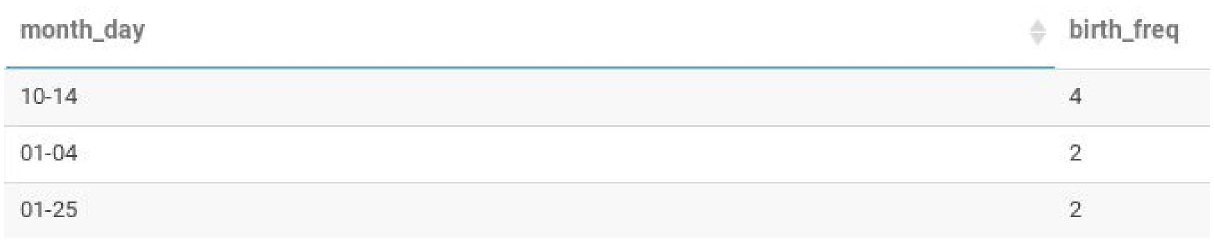
1.2. Create table in hive.

-- Executed in bash, hive shell or HUE
```
CREATE EXTERNAL TABLE IF NOT EXISTS birthday (
  id INT,
  birth_date STRING
)
COMMENT 'Birthdays table'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/cloudera/orange/birthday'
```

1.3. Hive

1.3.1. Generate a query to obtain a most birthdays on 1 day

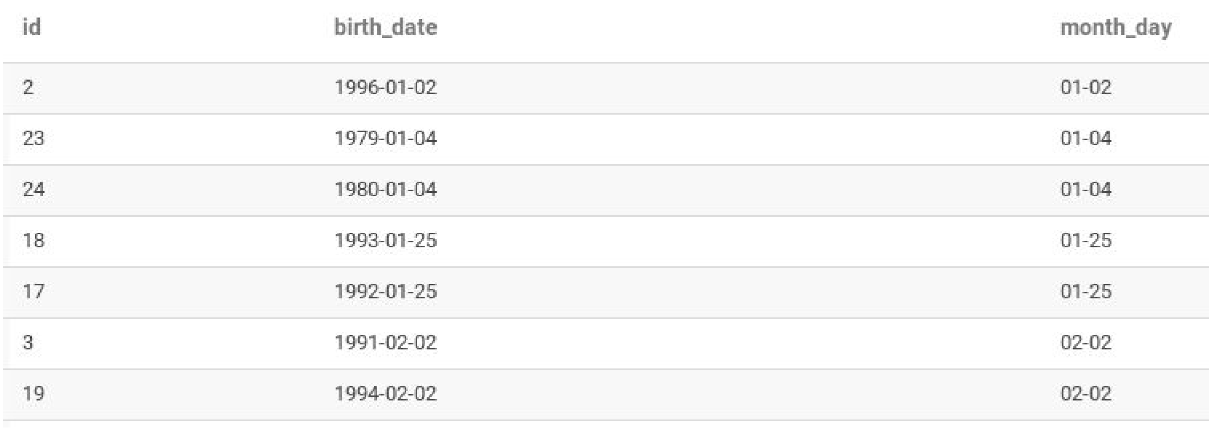
```
-- Calculate top three days of birth dates
WITH a AS (
       SELECT  CONCAT(SPLIT(birth_date,  '-')[1],  "-",  SPLIT(birth_date,
'-')[2]) AS month_day
   FROM birthday
)

SELECT month_day, COUNT(*) AS birth_freq
FROM a
WHERE month_day IS NOT NULL
GROUP BY month_day
ORDER BY birth_freq DESC, month_day ASC
LIMIT 3;
```



month_day	birth_freq
10-14	4
01-04	2
01-25	2

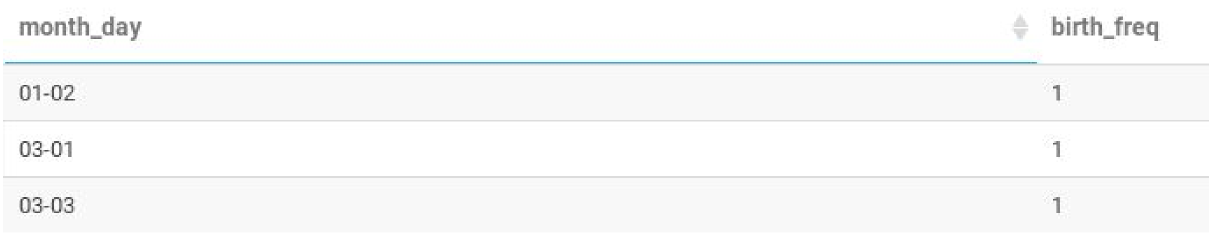
1.3.2. Generate a query to obtain Least birthdays are on 11 months

```
-- Birthdays in the next eleven months
SELECT id, birth_date,
  CONCAT(SPLIT(birth_date,  '-')[1],  "-",  SPLIT(birth_date,  '-')[2])  AS
month_day
FROM birthday
WHERE   CAST(datediff(add_months(to_date(from_unixtime(unix_timestamp())),
11), birth_date)/365.25 AS INT)
-                   CAST(datediff(to_date(from_unixtime(unix_timestamp())),
birth_date)/365.25 AS INT) <> 0
ORDER BY month_day
```



id	birth_date	month_day
2	1996-01-02	01-02
23	1979-01-04	01-04
24	1980-01-04	01-04
18	1993-01-25	01-25
17	1992-01-25	01-25
3	1991-02-02	02-02
19	1994-02-02	02-02

```
-- Grouped and sorted ascending by month and day (least frequencies)

WITH b AS (
    SELECT id, birth_date,
       CONCAT(SPLIT(birth_date, '-')[1], "-", SPLIT(birth_date, '-')[2]) AS
month_day
    FROM birthday
                                                                     WHERE
CAST(datediff(add_months(to_date(from_unixtime(unix_timestamp())),     11),
birth_date)/365.25 AS INT)
               -   CAST(datediff(to_date(from_unixtime(unix_timestamp())),
birth_date)/365.25 AS INT) <> 0
    ORDER BY month_day
)

SELECT month_day, COUNT(*) AS birth_freq
FROM b
GROUP BY month_day
ORDER BY birth_freq, month_day
LIMIT 3;
```



month_day	birth_freq
01-02	1
03-01	1
03-03	1

2. Scala.

2.1. Write functions to read and write (from hive and aws s3)

According to Spark documentation:
https://spark.apache.org/docs/2.4.0/sql-data-sources-hive-tables.html

"Configuration of Hive is done by placing your `hive-site.xml`, `core-site.xml` (for security configuration), and `hdfs-site.xml` (for HDFS configuration) file in `conf/`."

These files should be properly configured in the cluster where the Spark application runs. They are not required in a local test.

Reference for interacting with **AWS S3**:
Considerations with credentials:
https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/spark_s3.html

The following dependencies in build.sbt file are included:

```
"org.apache.hadoop" % "hadoop-aws"  % "3.2.1",
"com.amazonaws" % "aws-java-sdk" % "1.11.810"
```

Read and write to a local Hive warehouse is done through the main function developed.

2.2. Generate output columns in the result dataset with these changes:

- Apply previous hive query in functions and generate columns
- Email providers with more than 10k posts
- Post by email providers
- Year/s with max sign ups
- Class C IP address frequency by 1st octet
- Frequency of IP address based on first 3 octets
- Number of referral by members

As no dataset has been provided, some test records have been generated with the following schema:

```
email,numPosts,yearSignup,referred,ipAddress
aaaaa@gmail.com,10101,1990,m1|m2,10.0.0.1
```

- email (String): email of a person from a certain provider.
- numPosts (Integer): the number of posts done in the platform (e.g: Wordpress).
- yearSignup (Integer): the year when the user signed up in the platform.
- referred (String): the users which referred this one (e.g.: for being a good writer) separated by a pipe ('|').
- ipAddress (String): the last IP Address of the user when logged into the platform.

In this case email would eventually serve as the primary key for the table. Next there are the clarifications considered when developing the code:

- **Post by email providers:** initially a provider (String) field was considered. It contained an email provider company (e.g: Google, Hotmail, Yahoo, etc.).
  This was an initial assumption for a field that was discarded as grouping data depending on these values did not have much sense.
  Therefore, the group by developed in the "Post by email provider task" was commented and in a real case with records containing a collection of posts (e.g. the record contains post links), the resolution would be similar to the one proposed for "Number of referral by members".

2.3. Generate Scala Test for all code

ScalaTest is used:
https://www.scalatest.org/user_guide/using_scalatest_with_sbt

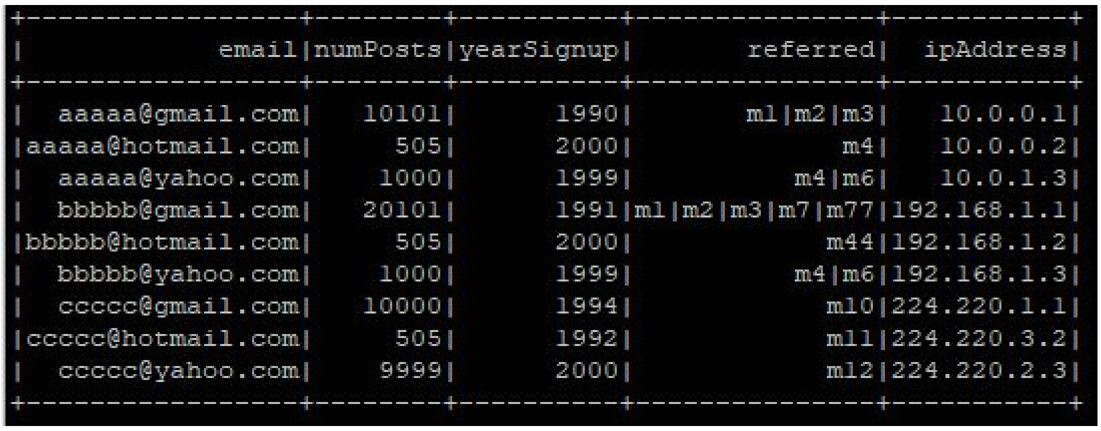
4. Workflow: Generate .jar using sbt or gradle.

sbt clean package

After installing locally: Spark 2.4.6, Scala 2.13.2 and sbt 1.3.12

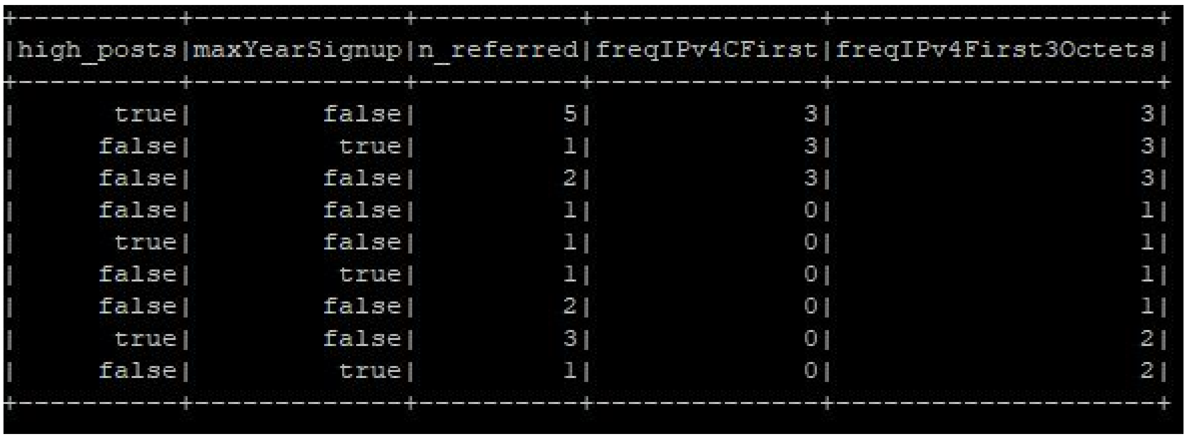
Execute the application with:

```
spark-submit --class com.orange.OrangeScala --master local[2]
target\scala-2.11\orange_scala_2.11-1.0.jar src\main\resources\users.csv
```



```
+-----------------+--------+----------+---------------+-----------+
|            email|numPosts|yearSignup|       referred|  ipAddress|
+-----------------+--------+----------+---------------+-----------+
|  aaaaa@gmail.com|   10101|      1990|       m1|m2|m3|   10.0.0.1|
|aaaaa@hotmail.com|     505|      2000|             m4|   10.0.0.2|
|  aaaaa@yahoo.com|    1000|      1999|          m4|m6|   10.0.1.3|
|  bbbbb@gmail.com|   20101|      1991|m1|m2|m3|m7|m77|192.168.1.1|
|bbbbb@hotmail.com|     505|      2000|            m44|192.168.1.2|
|  bbbbb@yahoo.com|    1000|      1999|          m4|m6|192.168.1.3|
|  ccccc@gmail.com|   10000|      1994|            m10|224.220.1.1|
|ccccc@hotmail.com|     505|      1992|            m11|224.220.3.2|
|  ccccc@yahoo.com|    9999|      2000|            m12|224.220.2.3|
+-----------------+--------+----------+---------------+-----------+
```

"users" table - initial state


```
+----------+-------------+----------+-------------+--------------------+
|high_posts|maxYearSignup|n_referred|freqIPv4CFirst|freqIPv4First3Octets|
+----------+-------------+----------+-------------+--------------------+
|      true|        false|         5|            3|                   3|
|     false|         true|         1|            3|                   3|
|     false|        false|         2|            3|                   3|
|     false|        false|         1|            0|                   1|
|      true|        false|         1|            0|                   1|
|     false|         true|         1|            0|                   1|
|     false|        false|         2|            0|                   1|
|      true|        false|         3|            0|                   2|
|     false|         true|         1|            0|                   2|
+----------+-------------+----------+-------------+--------------------+
```
“users_output” table - new columns added due to computations in Spark application

The CSV file stored with AwsOps -simulating an S3 write- contains:

```
email,numPosts,yearSignup,referred,ipAddress,high_posts,maxYearSignup,n_referred,freqIPv4CFirst,freqIPv4First3Octets
ccccc@hotmail.com,505,1992,m11,224.220.3.2,false,false,1,0,1
ccccc@gmail.com,10000,1994,m10,224.220.1.1,true,false,1,0,1
ccccc@yahoo.com,9999,2000,m12,224.220.2.3,false,true,1,0,1
aaaaa@gmail.com,10101,1990,m1|m2|m3,10.0.0.1,true,false,3,0,2
aaaaa@hotmail.com,505,2000,m4,10.0.0.2,false,true,1,0,2
aaaaa@yahoo.com,1000,1999,m4|m6,10.0.1.3,false,false,2,0,1
bbbbb@gmail.com,20101,1991,m1|m2|m3|m7|m77,192.168.1.1,true,false,5,3,3
bbbbb@hotmail.com,505,2000,m44,192.168.1.2,false,true,1,3,3
bbbbb@yahoo.com,1000,1999,m4|m6,192.168.1.3,false,false,2,3,3
```