4.3. 빅데이터 처리/탐색 실습하기

1) MapReduce 실습

실습1 MapReduce 단어 카운트 실습하기

STEP_1. MapReduce 자바 프로그래밍

ㄴ HDFS 입출력 프로그래밍

ㄴ MapReduce 프로그래밍 후 jar 라이브러리 파일 생성

STEP_2. Hadoopo 실행 및 sample 파일 준비

ㄴ Hadoop 실행(Namenode에서 실행)

```
#start-all.sh
#mr-jobhistory-daemon.sh start historyserver
```

ㄴ HDFS - MapReduce 디렉터리 생성

```
#hdfs dfs -mkdir /MapReduce
```

ㄴ WordCount sample.txt 파일 HDFS로 복사

```
#hdfs dfs -put /root/sample.txt /MapReduce
```

STEP_3. WordCount 프로그램 실행

ㄴ MapReduce 실행 명령어 입력

```
#yarn jar xxxx.jar sub2.WordCountMain /MapReduce/Sample.txt /MapReduce/output
```

ㄴ 실행 중인 MapReduce 작업 확인

```
#yar application -list
```

ㄴ http://192.168.xxx.101:8088에서 실행 중인 MapReduce 작업 확인

실습2 전국 날씨 평균 구하기 MapReduce 실습

2) Hive 실습

실습1 Hive 설치 및 설정 실습하기

STEP_1. Hive Metastore 데이터베이스 생성/계정 설정

ㄴ mysql(mariadb)가 설치되어 있어야 함(메뉴얼 참고)

```
#mysql -u root -p

mysql>CREATE DATABASE hive_metastore_db;
mysql>CREATE USER 'hive'@'localhost' IDENTIFIED BY '1234';
mysql>CREATE USER 'hive'@'%' IDENTIFIED BY '1234';
mysql>GRANT ALL ON *.* TO 'hive'@'localhost' IDENTIFIED BY '1234';
mysql>GRANT ALL ON *.* TO 'hive'@'%' IDENTIFIED BY '1234';
mysql>FLUSH PRIVILEGES;
mysql>exit
```

STEP_2. Hive용 HDFS 디렉터리 생성

ㄴ Hive 2.x 버전 다운로드/압축해제/이동/링크 생성(Namenode 실행)

```
#hdfs dfs -mkdir /hive
#hdfs dfs -mkdir /hive/warehouse
```

STEP_3. Hive 설치/설정

ㄴ Hive 2.x 버전 다운로드/압축해제/이동/링크 생성(Namenode 실행)

```
#wget http://mirror.apache-kr.org/hive/hive-2.x.x/apache-hive-2.x.x-bin.tar.gz
#tar xvfz apache-hive-2.x.x-bin.tar.gz
#mv apache-hive-2.x.x-bin /home/bigdata/
#cd /home/bigdata/
#ln -s apache-hive-2.3.7-bin/ hive
```

ㄴ Hive 환경설정(Namenode 실행)

```
#vi ~/.bashrc

// 맨 아래에 선언 추가
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export HIVE_HOME=/home/bigdata/hive
export PATH=$PATH:$HIVE_HOME/bin
```

ㄴ 현재 쉘에 반영(Namenode 실행)

```
#source ~/.bashrc
```

ㄴ Hive 설정 template 파일 복사(Namenode 실행)

```
#cd /home/bigdata/hive/conf
#cp hive-env.sh.template hive-env.sh
#cp hive-exec-log4j2.properties.template hive-exec-log4j2.properties
#cp hive-log4j2.properties.template hive-log4j2.properties
#cp hive-default.xml.template hive-site.xml
```

ㄴ hive-env.sh 설정(Namenode 실행)

```
#vi /home/bigdata/hive/conf/hive-env.sh

48라인 주석해제, Hadoop 경로 입력
HADOOP_HOME=/home/bigdata/hadoop
```

ㄴ hive-site.xml 설정(Namenode 실행)

```
#vi /home/bigdata/hive/conf/hive-site.xml

기존 내용 모두 삭제 후 아래 내용 입력, 빠른 삭제를 위해 vi명령 5000dd 입력
<configuration>
  <property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/hive/warehouse</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost:3306/hive_metastore_db?
                                        createDatabaseIfNotExist=true</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>1234</value>
  </property>
</configuration>
```

ㄴ http://dev.mysql.com/downloads/connector/j/ 에서 리눅스용 mysql-connector-java-5.x.tar.gz 파일 다운로드

ㄴ FileZilla FTP로 mysql-connector-java-5.x.tar.gz 파일을 /root 경로에 업로드

ㄴ mysql-connector-java-5.x.tar.gz 압축해제/라이브러리 복사

```
#tar zxvf mysql-connector-java-5.x.tar.gz
#cp mysql-conn...-java-5.x/mysql-connector-java-5.x-bin.jar $HIVE_HOME/lib
```

STEP_4. Metastore 설정

ㄴ metastore 초기화

```
#schematool -initSchema -dbType mysql

마지막 complated 확인 및 직접 metastore에 생성된 테이블 확인
…
Initialization script hive-schema-2.3.0.mysql.sql
Initialization script completed
schemaTool completed
```

ㄴ 만약 Metastore state would be inconsistent!! 에러 발생 시 데이터베이스 삭제 후 다시 생성

```
mysql>DROP DATABASE hive_metastore_db;
```

STEP_5. Hive 실행

ㄴ Hive 실행 후 Hive CLI 확인

```
#hive
…
hive>exit;
```

실습2 Hive 쿼리(HiveQL) 실습하기

STEP_1. HiveQL 실습

ㄴ 테이블 생성, 입력, 조회

```
hive>SHOW TABLES;

# Hive 내부 테이블
# Hive 기본 Warehouse에 테이블 파일이 생성되고, 데이터가 저장되는 Hive 기본 테이블
hive>CREATE TABLE `User1` (
    > `uid` String,
    > `name` String,
    > `hp` String,
    > `age` Int
    > );

# Hive 외부 테이블
# Hive 기본 Warehouse에 테이블 파일이 생성되지 않고, location 경로에 생성되는 테이블
hive>CREATE EXTERNAL TABLE `User2` (
    > `uid` String,
    > `name` String,
    > `hp` String,
    > `age` Int
    > )
    > location "/hive/external/User2";

hive>INSERT INTO User1 VALUES ('u101', '김유신', '010-1234-1111', 23);
hive>INSERT INTO User1 VALUES ('u102', '김춘추', '010-1234-2222', 21);
hive>INSERT INTO User1 VALUES ('u103', '장보고', '010-1234-3333', 27);

hive>INSERT INTO User2 VALUES ('u201', '강감찬', '010-1234-4444', 51);
hive>INSERT INTO User2 VALUES ('u202', '이순신', '010-1234-5555', 43);
hive>INSERT INTO User2 VALUES ('u203', '정약용', '010-1234-6666', 47);

hive>SELECT * FROM `User1`;
hive>SELECT * FROM `User2`;
```

STEP_2. 데이터 확인

ㄴ http://192.168.100.101:50070

ㄴ /hive/warehouse/user1 파일내용 확인

ㄴ /hive/external/user2 파일내용 확인

실습3 전국 날씨 평균 구하기 HiveQL 실습

3) Spark 실습

실습1 Spark 설치 및 설정 실습하기

STEP_1. Spark 설치/설정

ㄴ Spark 2.x.x 버전 다운로드/압축해제/이동/링크 생성(Namenode 실행)

```
#wget http://mirror.apache-kr.org/spark/spark-2.4.x/spark-2.4.x-bin-hadoop2.7.tgz
#tar xvfz spark-2.4.x-bin-hadoop2.7.tgz
#mv spark-2.4.x-bin-hadoop2.7 /home/bigdata/
#cd /home/bigdata/
#ln -s spark-2.4.x-bin-hadoop2.7/ spark
```

ㄴ Hive 환경설정(Namenode 실행)

```
#vi ~/.bashrc

// 맨 아래에 선언 추가
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export SPARK_HOME=/home/bigdata/spark
export PATH=$PATH:$SPARK_HOME/bin
```

ㄴ 현재 쉘에 반영(Namenode 실행)

```
#source ~/.bashrc
```

STEP_2. Spark 실행

ㄴ Spark shell 확인 후 종료

```
#spark-shell
Welcome to

      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.6
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_252)

Type in expressions to have them evaluated.

Type :help for more information.


scala>:q
```

실습2 Spark RDD 실습하기

STEP_1. RDD 기본 실습

ㄴ 기본 RDD, 외부파일을 이용한 RDD 생성

```
scala>val rdd1 = sc.parallelize(List("a", "b", "c", "d", "e"))
scala>val rdd2 = sc.parallelize(1 to 100, 10)
scala>val rdd3 = sc.textFile("/sample/animal.txt")
```

ㄴ RDD 데이터 출력, collect Action 사용

```
scala>var rdd1Result = rdd1.collect
scala>print(rdd1Result.mkString(", "))
scala>print(rdd2.collect.mkString(", "))
scala>print(rdd3.collect.mkString(", "))
```

ㄴ RDD 데이터 Count 출력

```
scala>val rdd1Cnt = rdd1.count
scala>print(rdd1Cnt)
scala>print(rdd2.count)
scala>print(rdd3.count)
```

STEP_2. RDD Transformation - map관련 API 실습

ㄴ map은 RDD에 속하는 모든 요소에 적용한 뒤 그 결과로 구성된 새로운 RDD 생성

```
scala>val rdd = sc.parallelize(1 to 5)
scala>val result = rdd.map(_+1)
scala>print(result.collect.mkString(", "))
```

ㄴ flatMap은 RDD에 속하는 모든 요소에 반복 적용한 뒤 그 결과로 구성된 새로운 RDD 생성

```
scala>val list = List("apple,orange", "tiger,lion,eagle,shark", "kim,lee,jeong")
scala>var rdd = sc.parallelize(list)
scala>var result1 = rdd.map(_.split(","))
scala>print(result1.collect.map(_.mkString("|")))

scala>var result2 = rdd.flatMap(_.split(","))
scala>print(result2.collect.map(_.mkString("|")))
```

STEP_3. RDD Transformation - 그룹관련 API 실습

ㄴ zip은 두 개의 서로 다른 RDD를 각 요소의 인덱스에 따라 하나의 (키, 값) 쌍으로 그룹화

```
scala>val rdd1 = sc.parallelize(List("a", "b", "c"))
scala>val rdd2 = sc.parallelize(List(1, 2, 3))
scala>val result = rdd1.zip(rdd2)
scala>print(result.collect.mkString(", "))
```

STEP_4. RDD Transformation - 집합관련 API 실습

ㄴ distinct는 RDD 원소에서 중복을 제외한 새로운 RDD를 생성

```
scala>val rdd = sc.parallelize(List(1, 2, 3, 1, 2, 3, 1, 2, 3, 4, 5))
scala>val result = rdd.distinct()
scala>print(result.collect.mkString(", "))
```

ㄴ subtract는 두 개의 RDD 원소에서 서로 같은 원소를 제외한 새로운 RDD생성

```
scala>val rdd1 = sc.parallelize(List("a", "b", "c", "d", "e"))
scala>val rdd2 = sc.parallelize(List("d", "e"))
scala>val result = rdd1.subtract(rdd2)
scala>print(result.collect.mkString(", "))
```

ㄴ union은 두 개의 RDD 원소를 요소로 새로운 RDD생성

```
scala>val rdd1 = sc.parallelize(List("a", "b", "c"))
scala>val rdd2 = sc.parallelize(List("d", "e", "f"))
scala>val result = rdd1.union(rdd2)
scala>print(result.collect.mkString(", "))
```

ㄴ join은 두 개의 서로 같은 키를 가진 요소를 그룹으로 새로운 RDD생성

```
scala>val rdd1 = sc.parallelize(List("a", "b", "c", "d", "e")).map((_, 1))
scala>val rdd2 = sc.parallelize(List("b", "c")).map((_, 2))
scala>val result = rdd1.join(rdd2)
scala>print(result.collect.mkString("\n"))
```

STEP_5. RDD Transformation - 집계관련 API 실습

ㄴ reduceByKey는 RDD 원소에서 같은 키를 갖는 값들을 하나로 병합해 새로운 RDD생성

```
scala>val rdd = sc.parallelize(List("a", "b", "b")).map((_, 1))
scala>val result = rdd.reduceByKey(_+_)
scala>print(result.collect.mkString(", "))
```

ㄴ filter는 RDD의 원소 중에서 원하는 요소만 남기고 새로운 RDD생성

```
scala>val rdd = sc.parallelize(1 to 10)
scala>val result = rdd.filter(_ > 5)
scala>print(result.collect.mkString(", "))
```

ㄴ sortByKey는 RDD의 원소를 키 값을 기준으로 정렬

```
scala>val rdd = sc.parallelize(List("q", "z", "a"))
scala>val result = rdd.map((_, 1)).sortByKey()
scala>print(result.collect.mkString(", "))
```

STEP_6. RDD Action - 출력관련 API 실습

ㄴ countByValue는 RDD의 원소의 나타나는 횟수를 맵 형태로 출력

```
scala>val rdd = sc.parallelize(List("a", "b", "a", "c", "d", "b", "a"))
scala>val result = rdd.countByValue
scala>print(result)
```

ㄴ foreach는 RDD의 모든 원소에 특정 함수를 적용

```
scala>val rdd = sc.parallelize(1 to 10)
scala>rdd.foreach { v =>
          println(s"rdd data : ${v}")
      }
```

ㄴ saveAsTextFile은 RDD의 요소를 파일로 저장

```
scala>val rdd = sc.parallelize(1 to 1000, 3)
scala>rdd.saveAsTextFile("/sample/rdd.txt")
```

실습3 Spark RDD API를 이용한 WordCount 실습하기

실습3 Spark DataFrame 실습하기

STEP_1. DataFrame 기본 실습

ㄴ 기본 DataFrame, 외부파일을 이용한 DataFrame 생성

```
scala>val df1 = spark.read.text("/sample/animal.txt")
scala>val df2 = spark.read.csv("/sample/user.csv")
scala>val userDF = spark.read.option("header", "true").csv("/sample/user.csv")
scala>val salesDF = spark.read.option("header", "true").csv("/sample/sales2017.csv")

# RDD를 DataFrame으로 변환
scala>val rdd = sc.textFile("/sample/animal.txt")
scala>val rddDF = rdd.toDF()

scala>df1.show()
scala>df2.show(100)
scala>userDF.show
scala>rddDF.show
```

STEP_2. DataFrame 연산

ㄴ select() 데이터프레임의 특정 컬럼을 갖는 새로운 데이터프레임 생성

```
scala>userDF.select("*").show
scala>userDF.select("uid", "name").show
scala>userDF.select('uid, 'name).show
scala>userDF.select('uid, 'name, 'age).show
```

ㄴ as() 데이터프레임의 특정 컬럼에 별칭

```
scala>userDF.select('name, 'age+1).show
scala>userDF.select('name, ('age+1).as("age")).show
```

ㄴ where() 데이터프레임의 특정조건을 만족하는 행 선택

```
scala>userDF.select("*").where('name === "홍길동").show
scala>userDF.where('name !== "홍길동").show
scala>userDF.where(col("age") > 30).show
scala>userDF.where('age > 30).show
scala>userDF.select('name, 'age).where('age > 30).show
```

ㄴ orderBy(), sort() 데이터프레임의 컬럼 정렬

```
scala>userDF.orderBy("age").show
scala>userDF.orderBy('age).show
scala>userDF.orderBy('age.asc).show
scala>userDF.orderBy('age.desc).show
scala>userDF.select('name, 'age).where('age > 30).orderBy('name.asc).show
scala>userDF.sort("age").show
scala>userDF.sort(asc("age")).show
scala>userDF.sort(desc("age")).show
scala>userDF.select('name, 'age).where('age > 30).sort("name").show
```

ㄴ count()는 데이터프레임의 전체 행의 개수를 출력

```
scala>userDF.count()
scala>userDF.where('age > 30).count
scala>userDF.where('age > 30).select(count("name")).show
```

ㄴ sum(), max(), mean(), describe() 데이터프레임의 합, 최대값, 평균, 기본통계 출력

```
scala>userDF.select(sum("age")).show
scala>userDF.select(max("age")).show
scala>userDF.select(mean("age")).show
scala>salesDF.describe("sale").show
```

ㄴ agg() 데이터프레임의 특정 컬럼에 대해 집계함수를 실행

```
scala>userDF.agg(max("age")).show
scala>userDF.agg(max("age"), min("age")).show
scala>salesDF.agg("sale"->"max").show
scala>salesDF.agg("sale"->"max", "sale"->"min", "sale"-> "sum", "sale"->"mean").show
```

ㄴ groupBy() 데이터프레임의 컬럼의 중복값을 그룹화

```
scala>userDF.groupBy("name").count.show
scala>userDF.where('age > 30).groupBy("name").count.show
scala>salesDF.groupBy("uid").agg("sale"->"sum").show
```

ㄴ join() 하나 이상의 데이터프레임을 병합

```
scala>userDF.join(saleDF, Seq("uid")).show
```

STEP_3. DataFrame 출력

ㄴ write() 데이터프레임을 파일로 저장

```
scala>val joinDF = userDF.join(saleDF, Seq("uid"))
scala>joinDF.write.format("com.databricks.spark.csv").save("/sample/joinDF")
```