Spark SQL

11기 박지원

INDEX

Spark SQL

데이터 & 데이터베이스

데이터

컴퓨터 안에 기록되어 있는 숫자

데이터베이스

넓은 의미 : 데이터의 집합

통용되는 의미 : 정리된 데이터

→ 데이터베이스 내의 데이터는 영구적으로 보존되어야 하기 때문에 하드디스크나 플래시메모리(SSD) 등 비휘발성 저장장치에 저장됨

DB & DBMS

DB

DataBase

DBMS

DataBase Management System → 데이터베이스 관리시스템

데이터베이스를 효율적으로 관리하는 소프트웨어

SQL

- DBMS(데이터베이스 관리 시스템)와의 대화에 필요한 것이 SQL

- 데이터베이스의 여러 중류 중에 관계형 데이터베이스 관리 시스템(RDBMS : Relational DataBase Management System)을 조작 할 때 사용한다

- IBM이 개발한 'SEQUEL'이라는 관계형 데이터베이스 조작용 언어를 기반으로 만들어짐

- ISO에 의해 표준화가 진행된 표준 언어 → 생산성을 향상시킬 수 있다

데이터베이스 종류

계층형

역사가 오래됨

폴더와 파일 등의 계층 구조로 데이터 저장하는 방식

현재는 많이 채택되지 않음

ex) 하드디스크나 DVD 파일시스템

관계형

'관계대수 (relational algebra)' 에 착안하여 고안

행과 열을 가지는 표 형식 데이터를 저장하는 형태

관계대수 자체는 표와 관련이 없지만 일단~

표는 2차원 데이터 (행x열)

여러 개의 표에 이름을 붙여 관리

이들은 SQL로 조작 가능

데이터베이스 종류

객체지향

'객체(object)' 를 중심으로 프로그래밍하는 언어
가능하면 객체 그대로를 데이터베이스의 데이터로 저장하는 것

XML

XML은 태그를 이용해 마크업 문서를 작성할 수 있게 정의한 것
HTML 태그와 흡사 → <data> </data>의 모양
XQuery 명령어 사용

키-밸류 스토어 (KVS)

키와 밸류의 조합으로 이루어진 데이터를 저장하는 데이터베이스
NoSQL (Not only SQL)이라는 슬로건에서 생겨남
열 지향 데이터베이스라고도 불림

간단한 SQL문

```
SELECT 열 FROM 테이블명;
```

- 테이블에서 특정 열을 보고싶을 때 쓰는 코드
- SQL 기본 코드

간단한 SQL문

모든 열

명령문의 마지막

```
SELECT * FROM sample21;
```

명령의 종류

테이블명

간단한 SQL문

```
SELECT 열 FROM 테이블명 WHERE 조건식;
```

- 조건식에 맞는 행을 테이블에서 뽑아낼 때 쓰는 코드
- 구의 순서는 무조건 SELECT → FROM → WHERE
- WHERE은 생략 가능 (모든 행이 검색 대상)

RDD

장점

- 분산환경에서 메모리 기반으로 빠르고 안정적으로 동작하는 프로그램 작성
- 풍부한 데이터 처리 연산

단점

- 메타데이터, “스키마”에 대해서 표현할 방법 부재

RDD

장점

- 분산환경에서 메모리 기반으로 빠르고 안정적으로 동작하는 프로그램 작성
- 풍부한 데이터 처리 연산

단점

- 메타데이터, "스키마"에 대해서 표현할 방법 부재

Spark SQL의 등장!

스키마를 표현하기 위해서 또 다른 유형의 데이터 모델과 API를 제공

Spark SQL

- SQL
- Dataset API

Spark SQL

Dataset API

~ Spark 1.6

DataFrame 클래스를
구현 언어와 상관없이
사용 가능

Spark 1.6

- Dataset의 등장

(자바, 스칼라에서만 사용 가능)

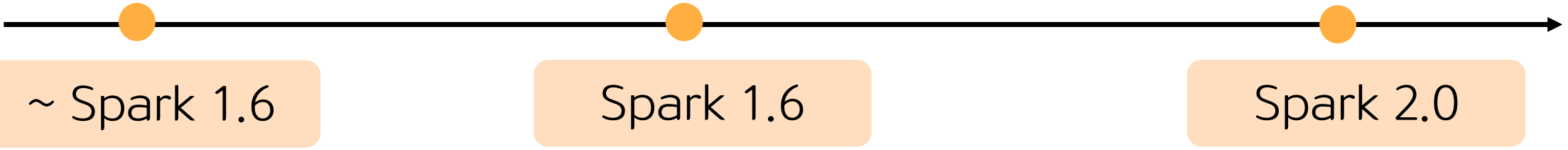
- DataFrame != Dataset
- 둘은 대체 관계가 아니고,
  자유롭게 선택하여 사용 가능

Spark 2.0

- DataFrame이 Dataset에 통합
- 타입 별칭(type alias) 기능이 있는 스칼라에서만
  DataFrame 사용 가능

Spark SQL

- Dataset API



	DataFrame	Dataset
Scala	O	O
Java	X	O
Python / R	O	X

Dataset

Dataset

	장점	단점
RDD	복잡한 코드 작성이 가능 컴파일 타임 오류 체크 가능	풍부하지 않은 API 상대적으로 낮은 성능
DataFrame	풍부한 API 옵티마이저를 기반으로 한 높은 성능	복잡한 코드 작성이 불가 컴파일 타임 오류 체크 불가

Dataset

	장점	단점
RDD	복잡한 코드 작성이 가능 컴파일 타임 오류 체크 가능	풍부하지 않은 API 상대적으로 낮은 성능
DataFrame	풍부한 API 옵티마이저를 기반으로 한 높은 성능	복잡한 코드 작성이 불가 컴파일 타임 오류 체크 불가

Dataset

연산의 종류와 주요 API

연산의 종류

Transformation

- 타입 연산 (typed operation)
- 비타입 연산 (untyped operation)

새로운 Dataset을 생성하는 연산
Action 연산이 호출될 때까지 수행되지 않음

Action

실제 데이터 처리를 수행하고 결과를 생성하는 연산

연산의 종류

```
scala > val data = 1 to 100 toList

scala > val ds = data.toDS

scala > val result = ds.map(_+1)

scala > ds.select(col("value")) + 1)
```

타입 연산
(typed operation)

- RDD의 Transformation 연산 중 하나인 map() 메서드 사용
- 타입은 Int

연산의 종류

```
scala > val data = 1 to 100 toList

scala > val ds = data.toDS

scala > val result = ds.map(_+1)

scala > ds.select(col("value")) + 1)
```

타입 연산
(typed operation)

```
scala> val result = ds.map(_+1)
result: org.apache.spark.sql.Dataset[Int] = [value: int]
```

```
scala> result.show()
+-----+
|value|
+-----+
|    2|
|    3|
|    4|
|    5|
|    6|
|    7|
|    8|
|    9|
|   10|
|   11|
|   12|
|   13|
|   14|
|   15|
|   16|
|   17|
|   18|
|   19|
|   20|
|   21|
+-----+
only showing top 20 rows
```

연산의 종류

```
scala > val data = 1 to 100 toList

scala > val ds = data.toDS

scala > val result = ds.map(_+1)

scala > ds.select(col("value")) + 1)
```

비타입 연산
(untyped operation)

- 데이터베이스의 table과 유사하게 처리
- 'value'라는 이름의 컬럼에 1을 더하고 select
- 타입은 org.apache.spark.sql.Column
- 본래의 타입이 아닌 Row와 Column 객체로 감싸서 처리하는 연산

연산의 종류

```
scala > val data = 1 to 100 toList

scala > val ds = data.toDS

scala > val result = ds.map(_+1)

scala > ds.select(col(“value”)) + 1)
```

비타입 연산
(untyped operation)

```
scala> ds.select(col("value")+1).show()
+-----------+
|(value + 1)|
+-----------+
|          2|
|          3|
|          4|
|          5|
|          6|
|          7|
|          8|
|          9|
|         10|
|         11|
|         12|
|         13|
|         14|
|         15|
|         16|
|         17|
|         18|
|         19|
|         20|
|         21|
+-----------+
only showing top 20 rows
```

코드 작성 절차 (Word Count)

코드 작성 절차 (Word Count)

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession₩
        .builder₩
        .appName("sample")₩
        .master("local[*]")₩
        .getOrCreate()

source = '''file:///home/ubuntu/18-2Engineering/
          Week05_181103/resources/countMe.txt'''
df = spark.read.text(source)

wordDF = df.select(explode(split(col("value"), " "))
                   .alias("word"))
result = wordDF.groupBy("word").count()
result.show()

outdir = '''file:///home/ubuntu/18-2Engineering/
          Week05_181103/output1/'''
result.write.format("csv").save(outdir)

spark.stop()
```

- Spark Session 생성
- DataFrame 생성
- 데이터 처리
- 처리된 결과를 외부 저장소에 저장
- Spark Session 종료

코드 작성 절차 (Word Count)

- Spark Session 생성

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession₩
        .builder₩
        .appName("sample")₩
        .master("local[*]")₩
        .getOrCreate()
```

코드 작성 절차 (Word Count)

- DataFrame 생성

```
source = '''file:///home/ubuntu/18-2Engineering/
            Week05_181103/resources/countMe.txt'''
df = spark.read.text(source)
```

코드 작성 절차 (Word Count)

- DataFrame 생성

```
source = '''file:///home/ubuntu/18-2Engineering/
            Week05_181103/resources/countMe.txt'''
df = spark.read.text(source)
```

df

```
+--------------------+
|               value|
+--------------------+
|A bad penny alway...|
|A barking dog nev...|
|A bird in the han...|
|A cat may look at...|
|A chain is only a...|
|A change is as go...|
|A dog is a man's ...|
|A drowning man wi...|
|A fish always rot...|
|A fool and his mo...|
|A friend in need ...|
|A golden key can ...|
|A good beginning ...|
|A good man is har...|
|A house divided a...|
|A person is known...|
|A house is not a ...|
|A journey of a th...|
|A leopard cannot ...|
|A little knowledg...|
+--------------------+
```

코드 작성 절차 (Word Count)

- 데이터 처리

```
wordDF = df.select(explode(split(col("value"), " "))
                   .alias("word"))
result = wordDF.groupBy("word").count()
result.show()
```

코드 작성 절차 (Word Count)

▲ 데이터 처리

```
                4       3      2          1
wordDF = df.select(explode(split(col("value"), " "))
                   5.alias("word"))
result = wordDF.groupBy("word").count()
result.show()
```

1. “value” 컬럼을
2. 띄어쓰기(“ ”) 기준으로 split 하고
3. 하나의 배열 컬럼에 포함된 단어들을 여러 개의 행으로 바꿔
4. 이 결과를 보여주는데
5. 이 때 보여주는 결과의 열 이름을 “word”라고 한다

코드 작성 절차 (Word Count)

- 데이터 처리

```
wordDF = df.select(explode(split(col("value"), " "))
                   .alias("word"))
result = wordDF.groupBy("word").count()
result.show()
```

wordDF

```
+-------+
|   word|
+-------+
|      A|
|    bad|
|  penny|
| always|
|  turns|
|     up|
|      A|
|barking|
|    dog|
|  never|
|  bites|
|      A|
|   bird|
|     in|
|    the|
|   hand|
|     is|
|  worth|
|    two|
|     in|
+-------+
```

result

```
+---------+-----+
|     word|count|
+---------+-----+
|     July|    1|
|    those|    5|
|    spoil|    3|
|   travel|    2|
|      few|    1|
|pack-drill|    1|
|   waters|    1|
|   harder|    1|
|     hope|    1|
|     some|    1|
|   taking|    1|
|  Sabbath|    1|
|    parts|    1|
|     lies|    1|
|   Mighty|    1|
| Tomorrow|    2|
|  vinegar|    1|
|  stomach|    2|
|  showers|    2|
|  flowers|    2|
+---------+-----+
```

코드 작성 절차 (Word Count)

- 처리된 결과를 외부 저장소에 저장

```
outdir = '''file:///home/ubuntu/18-2Engineering/
            Week05_181103/output1/'''
result.write.format("csv").save(outdir)
```

코드 작성 절차 (Word Count)

- Spark Session 종료

```
spark.stop()
```

Spark Session

Spark Session

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession₩
        .builder₩
        .appName("sample")₩
        .master("local[*]")₩
        .getOrCreate()
```

- 사용 목적
  - ✓ DataFrame / Dataset 정의
  - ✓ 사용자 정의 함수(UDF) 등록
- Spark SQL 프로그램은 가장 먼저 Spark Session을 생성해야 함

DataFrame 생성

DataFrame

- Spark SQL에서 사용하는 분산 데이터 모델
- Spark 2.0 미만에선 별도의 class를 가리키는 용어
- Spark 2.0 이후엔 Spark SQL의 또 다른 데이터 모델인 Dataset과 통합되면서 org.apache.spark.sql.Row 타입의 요소를 가진 Dataset을 가리키는 별칭(alias)이 됨
- DB의 table처럼 Row, Column의 구조를 가짐
- SQL문을 사용, Dataframe이 제공하는 API 이용
- 데이터에 대한 스키마 정보까지 함께 다룸 (cf. RDD only 데이터 값)

외부 데이터 소스로부터 DataFrame 생성

```
commonDir = "file:///home/ubuntu/18-2Engineering/Week05_181103/"
df1 = spark.read.json(commonDir + "./resources/people.json")
df2 = spark.read.parquet(commonDir + "./resources/users.parquet")
df3 = spark.read.text(commonDir + "./resources/people.txt")

df1.show()
df2.show()
df3.show()
```

df1

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

df2

```
+------+--------------+----------------+
|  name|favorite_color|favorite_numbers|
+------+--------------+----------------+
|Alyssa|          null|  [3, 9, 15, 20]|
|   Ben|           red|              []|
+------+--------------+----------------+
```

df3

```
+-----------+
|      value|
+-----------+
|Michael, 29|
|   Andy, 30|
| Justin, 19|
+-----------+
```

RDD 및 Collection으로 DataFrame 생성

	Name	Age	Job
Row 1	hayoon	7	student
Row 2	sunwoo	13	student
Row 3	hajoo	5	kindergartener
Row 4	jinwoo	13	student

RDD 및 Collection으로 DataFrame 생성

Row

	Name	Age	Job
Row 1	hayoon	7	student
Row 2	sunwoo	13	student
Row 3	hajoo	5	kindergartener
Row 4	jinwoo	13	student

Column

RDD 및 Collection으로 DataFrame 생성

```
row1 = Row(name="hayoon", age=7, job="student")
row2 = Row(name="sunwoo", age=13, job="student")
row3 = Row(name="hajoo", age=5, job="kindergartener")
row4 = Row(name="jinwoo", age=13, job="student")

data = [row1, row2, row3, row4]
rdd = spark.sparkContext.parallelize(data)

df4 = spark.createDataFrame(data)
df4.show()
```

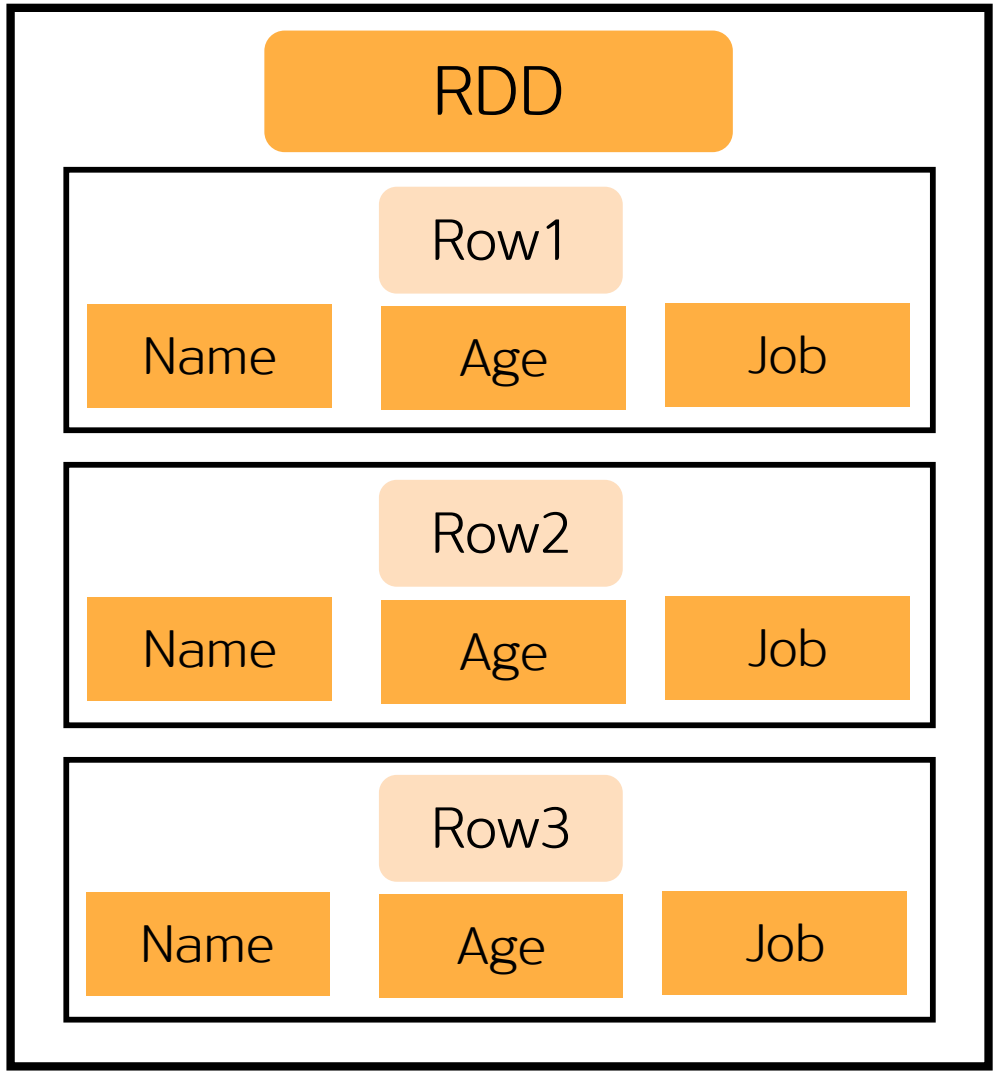
	Name	Age	Job
Row 1	hayoon	7	student
Row 2	sunwoo	13	student
Row 3	hajoo	5	kindergartener
Row 4	jinwoo	13	student

RDD 및 Collection으로 DataFrame 생성

```
row1 = Row(name="hayoon", age=7, job="student")
row2 = Row(name="sunwoo", age=13, job="student")
row3 = Row(name="hajoo", age=5, job="kindergartener")
row4 = Row(name="jinwoo", age=13, job="student")

data = [row1, row2, row3, row4]
rdd = spark.sparkContext.parallelize(data)

df4 = spark.createDataFrame(data)
df4.show()
```



RDD를 비롯해 Row, Column 형태로 만들 수 있는

Collection 객체만 있다면 Dataframe 생성 가능

List로 DataFrame 생성

```
row1 = Row(name="hayoon", age=7, job="student")
row2 = Row(name="sunwoo", age=13, job="student")
row3 = Row(name="hajoo", age=5, job="kindergartener")
row4 = Row(name="jinwoo", age=13, job="student")

data = [row1, row2, row3, row4]

df5 = spark.createDataFrame(data)
df5.show()
```

	Name	Age	Job
Row 1	hayoon	7	student
Row 2	sunwoo	13	student
Row 3	hajoo	5	kindergartener
Row 4	jinwoo	13	student

Schema 지정으로 DataFrame 생성

Schema

- 계획이나 도식
- 자료의 구조 및 표현 방법

Schema 지정으로 DataFrame 생성

Schema

- 계획이나 도식
- 자료의 구조 및 표현 방법

	Name	Age	Job
Row 1	hayoon	7	student
Row 2	sunwoo	13	student
Row 3	hajoo	5	kindergartener
Row 4	jinwoo	13	student

StructField

: 컬럼의 이름, 타입, null 허용 여부 지정

Schema 지정으로 DataFrame 생성

Schema

- 계획이나 도식
- 자료의 구조 및 표현 방법

	Name	Age	Job
Row 1	hayoon	7	student
Row 2	sunwoo	13	student
Row 3	hajoo	5	kindergartener
Row 4	jinwoo	13	student

StructType

: 컬럼으로 사용할 StructField의 목록 지정

Schema 지정으로 DataFrame 생성

```
sf1 = StructField("name", StringType(), True)
sf2 = StructField("age", IntegerType(), True)
sf3 = StructField("job", StringType(), True)

schema = StructType([sf1, sf2, sf3])

r1 = Row(name="hayoon", age=7, job="student")
r2 = Row(name="sunwoo", age=13, job="student")
r3 = Row(name="hajoo", age=5, job="kindergartener")
r4 = Row(name="jinwoo", age=13, job="student")
rows = [r1, r2, r3, r4]

df6 = spark.createDataFrame(rows, schema)
df6.show()
```

StructField(“컬럼명”, 데이터타입, null값 허용 여부)

Schema 지정으로 DataFrame 생성

```
sf1 = StructField("name", StringType(), True)
sf2 = StructField("age", IntegerType(), True)
sf3 = StructField("job", StringType(), True)

schema = StructType([sf1, sf2, sf3])

r1 = Row(name="hayoon", age=7, job="student")
r2 = Row(name="sunwoo", age=13, job="student")
r3 = Row(name="hajoo", age=5, job="kindergartener")
r4 = Row(name="jinwoo", age=13, job="student")
rows = [r1, r2, r3, r4]

df6 = spark.createDataFrame(rows, schema)
df6.show()
```

	Name	Age	Job
Row 1	hayoon	7	student
Row 2	sunwoo	13	student
Row 3	hajoo	5	kindergartener
Row 4	jinwoo	13	student

pandas 연동하여 DataFrame 생성

```
import pandas as pd
from pyspark.sql.functions import pandas_udf, PandasUDFType
spark.conf.set("spark.sql.execution.arrow.enabled", "true")
sales_data = {'name' : ['store2', 'store2', 'store1', 'store1'],
              'product' : ['note', 'bag', 'note', 'pen'],
              'amount' : [20, 10, 15, 20],
              'price' : [2000, 5000, 1000, 5000]}
pdf = pd.DataFrame(sales_data)
pdf
```

	amount	name	price	product
0	20	store2	2000	note
1	10	store2	5000	bag
2	15	store1	1000	note
3	20	store1	5000	pen

```
df = spark.createDataFrame(pdf)
df.show()
```

```
+------+------+-----+-------+
|amount|  name|price|product|
+------+------+-----+-------+
|    20|store2| 2000|   note|
|    10|store2| 5000|    bag|
|    15|store1| 1000|   note|
|    20|store1| 5000|    pen|
+------+------+-----+-------+
```

DataFrame 주요 연산

연산의 종류

Transformation

- 타입 연산 (typed operation)
- 비타입 연산 (untyped operation)

새로운 Dataset을 생성하는 연산
Action 연산이 호출될 때까지 수행되지 않음

Action

실제 데이터 처리를 수행하고 결과를 생성하는 연산

연산의 종류

Transformation

타입 연산
(typed operation)

DataFrame이 아닌 Dataset에서만 사용 가능

비타입 연산
(untyped operation)

Dataset의 구성요소가
org.apache.spark.sql.Row 타입인 경우
즉, DataFrame인 경우에만 사용 가능

Action

DataFrame 주요 연산

	연산	의미
액션 연산	show()	데이터셋에 저장된 데이터를 화면에 출력
	head(), first()	데이터셋의 첫 번째 row를 돌려줌
	take()	데이터셋의 첫 n개의 row를 돌려줌
	count()	데이터셋에 포함된 row의 개수를 리턴
	collect()	데이터셋에 포함된 모든 데이터를 로컬 컬렉션(배열, 리스트) 형태로 돌려줌
	describe()	숫자형 colum에 대한 기초 통계값을 포함하는 dataframe 생성
기본 연산	cache(), persist()	작업 중인 데이터를 메모리에 저장
	printSchema(), columns, dtypes, schema	스키마 정보를 조회
	createOrReplaceTempView()	DataFrame을 table로 변환
	explain()	DataFrame 처리와 관련된 실행 계획 정보를 출력

DataFrame 주요 연산

	연산	의미
비타입 트랜스포메이션	alias(), as()	컬럼 명에 원하는 이름을 붙일 수 있다
	isin()	컬럼의 값이 인자로 지정된 값에 포함되어 있는지 여부 확인
	max(), mean()	최대값, 평균값
	collect_list(), collect_set()	특정 컬럼 값을 모아서 하나의 리스트 또는 세트로 된 컬럼을 생성함
	count(), countDistinct()	세기, 중복을 제외하고 세기
	agg()	특정 컬럼에 대해 sum(), max()와 같은 집한 연산 수행
	groupBy()	same as SQL, Pandas groupBy()
	cube()	인자로 지정한 컬럼으로 구성된 큐브 생성
	intersect()	두 개의 DataFrame에 모두 속하는 row로만 구성된 DataFrame 생성
	join()	same as RDD, Pandas join()
	crossJoin()	Cartesian products 실행

~ 실습 ~

감사합니다