DataFrame

순천향대학교 컴퓨터공학과

이 상 정

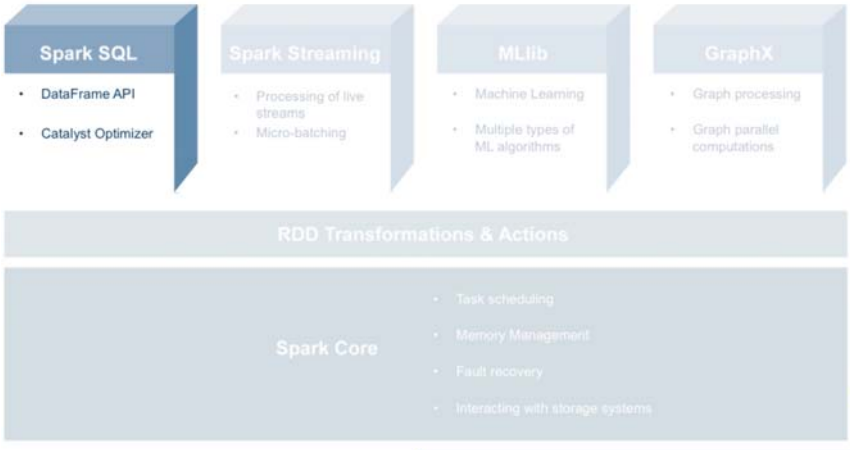
학습 내용

1. 데이터프레임 생성
2. 데이터프레임 연산
3. 사용자 정의 함수 생성

1. 데이터프레임 생성

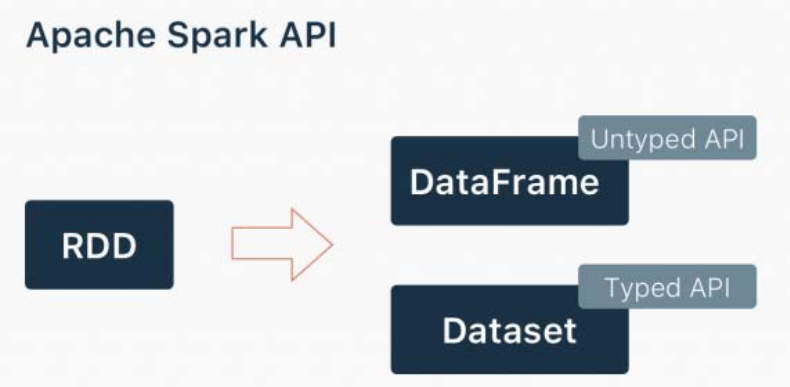
SparkSQL

- SparkSQL은 상호작용 셸, JDBC/ODBC, 데이터프레임 API를 사용하여 SQL 인터페이스 제공
  - 데이터프레임은 카탈리스트 최적화기(Catalyst optimizer)라는 관계형 최적화기를 사용
    - 데이터프레임을 사용한 SQL 질의는 최적화되어 일련의 스파크 작업들로 실행



복습 – RDD, 데이터세트, 데이터프레임 (1)

- 스파크는 다음 3가지 데이터의 추상화 제공
  - RDD (Resilient Distributed Dataset)
  - 데이터세트 (Dataset)
  - 데이터프레임 (DataFrame)



복습 – RDD, 데이터세트, 데이터프레임 (2)

- RDD
  - 비구조화된 데이터 (unstructured data)
    - 바이너리 스트림 (미디어), 텍스트 스트림
  - 타입이 정의되지 않은 데이터로 프로그래머가 최적화 수행
    - 낮은 성능으로 인하여 데이터 처리 속도를 높이는 최적화 필요
  - 데이터세트를 프로그래머가 조작하고, 저수준 변환 및 액션 사용
- 데이터세트
  - 반-구조화/구조화된 데이터 (semi-structured/structured data)
    - 구조화된 데이터: RDBMS, 반구조화된 데이터: json, csv
  - 기존의 최적화가 가능한 타입이 정의된 데이터
  - RDD에 Spark SQL의 프로그래밍 추상화 추가
    - Spark SQL은 구조화된 데이터 상의 작업을 수행
  - 적정 수준의 최적화 수행
    - RDD 보다는 높고, 데이터프레임보다는 낮은 최적화

복습 – RDD, 데이터세트, 데이터프레임 (3)

❑ 데이터프레임

- 반-구조화/구조화된 데이터 (semi-structured/structured data)
- 이름을 갖는 열(named column)들로 구성된 데이터세트
  - 개념적으로 관계형 데이터베이스의 테이블, 파이썬의 데이터 프레임과 유사
- 스파크 SQL 최적화 실행 엔진을 사용하여 높은 최적화 수행
- 다양한 데이터 형식과 저장 시스템 지원
  - 구조화된 데이터 파일, Hive 테이블, 외부 데이터베이스, RDD 로부터 구성 가능
- Scala, Python, Java, SparkR API 지원

데이터프레임 생성 소스

❑ 데이터프레임의 생성 시 소스

- 기존의 RDD
- 다른 데이터 소스

Data Sources

- Parquet
- JSON
- Hive tables
- Relational Databases

SparkSession

- 데이터프레임을 생성하기 위해서는 SparkSession이 필요

```
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().appName("AppName")
                                  . Config("spark.master", "yarn")
                                  .getOrCreate()
```

- 스파크 셸에서 SparkSession 이 이미 생성되어 있음

기존 RDD 소스에서 데이터프레임 생성

- 기존의 RDD에서 2가지 방식의 데이터프레임을 생성 방법

1. 기존 RDD를 묵시적으로 반영하여 생성(reflection)
   - 케이스 클래스를 사용하여 RDD를 변환
   - 스키마가 미리 알려진 경우 사용
2. 기존 RDD의 스키마를 프로그램으로 기술하여 생성 (programatic)
   - 열과 이들 타입들이 실행 시까지 알려지지 않는 경우 데이터프레임 구성 시 사용

묵시적으로 스키마 추론 – 케이스 클래스 (Case Class)

- 스칼라의 케이스 클래스는 쉽게 클래스를 정의하고 인스턴스를 생성하고, 패턴 매치
  - Java의 POJO(plain old java object)와 java beans와 유사

- 케이스 클래스를 사용하여 테이블의 스키마를 정의
  - 케이스 클래스로 전달되는 인수들이 테이블의 열이 됨
  - 인수의 이름과 타입이 테이블 열의 이름과 타입
  - 케이스 클래스는 중첩될 수도 있고, 시퀀스나 배열 같은 복잡한 데이터도 표현

묵시적으로 스키마 추론 절차

1. 필요한 클래스들 임포트
2. RDD 생성
3. 케이스 클래스 정의
4. RDD를 케이스 클래스 객체의 RDD로 변환
5. 암묵적으로 케이스 클래스의 RDD를 데이터프레임으로 변환
   - 변환된 데이터프레임에 연산과 함수 적용
6. 데이터프레임을 뷰(테이블)로 등록
   - 테이블 상에서 SQL 실행

SFPD 예 – 묵시적 스키마 추론 (1)

1. 필요한 클래스들 임포트

import org.apache.spark.sql.SparkSession

val spark: SparkSession = SparkSession.builder().getOrCreate()

// for implicit conversions from Spark RDD to Dataframe

import spark.implicits._

2. RDD 생성

val sfpdRDD = spark.sparkContext.
textFile("/sparkdata/sfpd/sfpd.csv").map(line=>line.split(","))

```
scala> import spark.implicits._
import spark.implicits._

scala> val sfpdRDD = spark.sparkContext.textFile("/sparkdata/sfpd/sfpd.csv").map(line
=>line.split(","))
sfpdRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[30] at map at <co
nsole>:29
```


SFPD 예 – 묵시적 스키마 추론 (2)

3. 케이스 클래스 정의

// Defining the Incidents case classcase class
Incidents(incidentnum:String, category:String, description:String, dayofweek:String, date:String, time:String, pddistrict:String, resolution:String, address:String, X:Float, Y:Float, pdid:String)

4. RDD를 케이스 클래스 객체의 RDD로 변환

// Mapping sfpdRDD to the case class

val sfpdCase = sfpdRDD.map(inc=>Incidents(inc(0),inc(1), inc(2), inc(3),inc(4),inc(5),inc(6),inc(7),inc(8),inc(9).toFloat,inc(10).toFloat , inc(11)))

```
scala> case class Incidents(incidentnum:String, category:String, descript
ion:String, dayofweek:String, date:String, time:String, pddistrict:String
, resolution:String, address:String, X:Float, Y:Float, pdid:String)
defined class Incidents

scala> val sfpdCase=sfpdRDD.map(inc=>Incidents(inc(0),inc(1), inc(2),inc(
3),inc(4),inc(5),inc(6),inc(7),inc(8),inc(9).toFloat,inc(10).toFloat, inc
(11)))
sfpdCase: org.apache.spark.rdd.RDD[Incidents] = MapPartitionsRDD[3] at ma
p at <console>:31
```

SFPD 예 – 묵시적 스키마 추론 (3)

5. 암묵적으로 케이스 클래스의 RDD를 데이터프레임으로 변환

   // converting sfpdRDD to a DataFrame

   val sfpdDF=sfpdCase.toDF()

6. 데이터프레임을 뷰(테이블)로 등록

   // Registering the sfpdDF as a temporary view

   sfpdDF.createOrReplaceTempView("sfpd")

```
scala> val sfpdDF=sfpdCase.toDF()
sfpdDF: org.apache.spark.sql.DataFrame = [incidentnum: string, category:
string ... 10 more fields]

scala> sfpdDF.createOrReplaceTempView("sfpd")
```


프로그램으로 스키마 구성 (1)

- 샘플 데이터 예
  - 아래의 1,3,5 번째 열의 데이터만 적재 사용

Sample Data

150599321 Thursday 7/9/15 23:45 CENTRAL

156168837 Thursday 7/9/15 23:45 CENTRAL

150599321 Thursday 7/9/15 23:45 CENTRAL

1. 입력 RDD로부터 Row RDD 생성

```
val rowRDD = sc.textFile("/user/user01/data/test.txt")
.map(x=>x.split(" "))
.map(p=>Row(p(0),p(2),p(4)))
```

프로그램으로 스키마 구성 (2)

2. StructType 객체 사용하여 스키마 정의
   - StructField 객체의 인수로 name, datatype, nullable 여부 지정

```
val testsch = StructType(Array(StructField("IncNum",
StringType,true), StructField("Date",StringType,true),
StructField("District",StringType,true)))
```

3. 데이터프레임 생성

```
val testDF = spark.createDataFrame(rowRDD, testsch)
```

   - 데이터프레임을 뷰로 등록

```
testDF.createOrReplaceTempView("test")
val incs = spark.sql("SELECT * FROM test")
```


데이터 소스로부터 데이터프레임 생성 (1)

- load() 메서드를 사용하여 데이터프레임 생성
  - 디폴트 소스 형식은 parquet (컬럼 기반 저장)

```
val usersDF =
        spark.read.load("examples/src/main/resources/users.parquet")
usersDF.select("name", "favorite_color").write.
                                    save ("namesAndFavColors.parquet")
```

  - 형식 지정

```
val peopleDF = spark.read.format("json").
                        load("examples/src/main/resources/people.json")
peopleDF.select("name", "age").write.format("parquet").
                                    save("namesAndAges.parquet")
```

데이터 소스로부터 데이터프레임 생성 (2)

- 특정 데이터 소스의 메서드 호출하여 데이터프레임 생성
  - JSON

```
val peopleDF =
spark.read.json("examples/src/main/resources/people.json")
```

  - JDBC

```
val connectionProperties = new Properties()
connectionProperties.put("user", "username")
connectionProperties.put("password", "password")
val jdbcDF = spark.read.jdbc("jdbc:postgresql:dbserver",
                             "schema.tablename", connectionProperties)
```


2. 데이터프레임 연산

SFPD 데이터 탐색 예

- 가장 사건이 많이 발생한 5개의 주소(address)는?
- 가장 사건이 많이 발생한 5개의 지구대(district)는?
- 가장 많은 10개의 사건 해결 유형(resolution)은?
- 가장 많은 3개의 범죄 유형(category)은?

데이터프레임 연산 유형

- 데이터프레임 상에서 수행되는 연산의 유형
  - 데이터프레임 액션
  - 데이터프레임 함수
  - 프로그래밍 언어와 결합된 질의
  - 일부 RDD 연산
  - 데이터프레임을 테이블 또는 파일로 출력

데이터프레임 주요 액션

Action	Description
`collect()`	Returns array containing all rows in DataFrame
`count()`	Returns number of rows in DataFrame
`describe(cols:String*)`	Computes statistics for numeric columns (count, mean, stddev, min and max)

데이터프레임 주요 함수

Function	Description
`cache()`	Cache this DataFrame
`columns`	Returns an array of all column names
`printSchema()`	Prints schema to console in tree format

프로그래밍 언어와 결합된 질의

L I Query	Description
`agg(expr, exprs)`	Aggregates on entire DataFrame
`distinct`	Returns new DataFrame with unique rows
`except(other)`	Returns new DataFrame with rows from this DataFrame not in other DataFrame
`filter(expr)`	Filter based on the SQL expression or condition

SFPD 예 – Top 5 주소 (1)

- 가장 사건이 많이 발생한 5개의 주소(address)는?

```
// Create a DataFrame by grouping the incidents by address.
val incByAdd = sfpdDF.groupBy("address")
// Count the number of incidents for each address
val numAdd = incByAdd.count
// Sort the result of previous step in descending order
val numAddDesc = numAdd.sort($"count".desc)
// Show the first five which is the top five addresses with the most
// incidents
val top5Add = numAddDesc
top5Add.show(5)
```

SFPD 예 – Top 5 주소 (2)

- 가장 사건이 많이 발생한 5개의 주소(address)는?

// Which are the five addresses with the most number of incidents?

```
val incByAdd = sfpdDF.groupBy("address")
                     .count.sort($"count".desc)
incByAdd.show(5)
```

```
scala> val incByAdd = sfpdDF.groupBy("address").count.sort($"count".desc)
incByAdd: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [address: string, c
ount: bigint]

scala> incByAdd.show(5)
+--------------------+-----+
|             address|count|
+--------------------+-----+
|800_Block_of_BRYA...|10852|
|800_Block_of_MARK...| 3671|
|1000_Block_of_POT...| 2027|
|2000_Block_of_MIS...| 1585|
|  16TH_ST/MISSION_ST| 1512|
+--------------------+-----+
only showing top 5 rows
```


SFPD 예 – Top 5 주소, SQL 버전

- SQL 버전으로 Top 5 주소 실행
  - SparkSession.sql 사용 ,
  - sfpd 뷰로 등록

// Top 5 addresses with most incidents: SQL

```
val topByAddressSQL = spark.sql("SELECT address,
    count(incidentnum) AS inccount FROM sfpd GROUP BY address
    ORDER BY inccount DESC LIMIT 5")
topByAddressSQL.show
```

```
scala> val topByAddressSQL = spark.sql("SELECT address, count(incidentnum) AS inccount
FROM sfpd GROUP BY address ORDER BY inccount DESC LIMIT 5")
topByAddressSQL: org.apache.spark.sql.DataFrame = [address: string, inccount: bigint]

scala> topByAddressSQL.show
+--------------------+--------+
|             address|inccount|
+--------------------+--------+
|800_Block_of_BRYA...|   10852|
|800_Block_of_MARK...|    3671|
|1000_Block_of_POT...|    2027|
|2000_Block_of_MIS...|    1585|
|  16TH_ST/MISSION_ST|    1512|
+--------------------+--------+
```

SFPD 예 – Top 5 지구대

- 가장 사건이 많이 발생한 5개의 지구대(district)는?

```
val incByDist = sfpdDF.groupBy("pddistrict")
                      .count.sort($"count".desc)
incByDist.show(5)
```

```
scala> val incByDist = sfpdDF.groupBy("pddistrict").count.sort($"count".desc)
incByDist: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [pddistrict: string
, count: bigint]

scala> incByDist.show(5)
+----------+-----+
|pddistrict|count|
+----------+-----+
|  SOUTHERN|73308|
|   MISSION|50164|
|  NORTHERN|46877|
|   CENTRAL|41914|
|   BAYVIEW|36111|
+----------+-----+
only showing top 5 rows
```


SFPD 예 – Top 5 지구대, SQL 버전

- 가장 사건이 많이 발생한 5개의 지구대(district)는?

```
// Top 5 districts with most incidents: SQL
val topByDistSQL = spark.sql("SELECT pddistrict,
   count(incidentnum) AS inccount FROM sfpd GROUP BY pddistrict
   ORDER BY inccount DESC LIMIT 5")
topByDistSQL.show
```

```
scala> val topByDistSQL = spark.sql("SELECT pddistrict, count(incidentnum) AS inccount
FROM sfpd GROUP BY pddistrict ORDER BY inccount DESC LIMIT 5")
topByDistSQL: org.apache.spark.sql.DataFrame = [pddistrict: string, inccount: bigint]

scala> topByDistSQL.show
+----------+--------+
|pddistrict|inccount|
+----------+--------+
|  SOUTHERN|   73308|
|   MISSION|   50164|
|  NORTHERN|   46877|
|   CENTRAL|   41914|
|   BAYVIEW|   36111|
+----------+--------+
```

SFPD 예 – Top 10 사건 해결

- 가장 많은 10개의 사건 해결 유형(resolution)은?

```
// Top 10 resolutions
val top10Res = sfpdDF.groupBy("resolution")
                     .count.sort($"count".desc)
top10Res.show(10)
```

```
scala> val top10Res = sfpdDF.groupBy("resolution").count.sort($"count".desc)
top10Res: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [resolution: string,
 count: bigint]

scala> top10Res.show(10)
+--------------------+------+
|          resolution| count|
+--------------------+------+
|                NONE|243538|
|       ARREST/BOOKED| 86766|
|        ARREST/CITED| 22925|
|   PSYCHOPATHIC_CASE|  8344|
|             LOCATED|  6878|
|           UNFOUNDED|  4551|
|COMPLAINANT_REFUS...|  4215|
|     JUVENILE_BOOKED|  2381|
|EXCEPTIONAL_CLEAR...|  1134|
|      JUVENILE_CITED|  1039|
+--------------------+------+
only showing top 10 rows
```


SFPD 예 – Top 10 사건 해결, SQL 버전

- 가장 많은 10개의 사건 해결 유형(resolution)은?

```
// Top 10 resolutions : SQL
val top10ResSQL = spark.sql("SELECT resolution,
  count(incidentnum) AS inccount FROM sfpd GROUP BY resolution
  ORDER BY inccount DESC LIMIT 10")
top10ResSQL.show
```

```
scala> val top10ResSQL = spark.sql("SELECT resolution, count(incidentnum) AS inccount F
ROM sfpd GROUP BY resolution ORDER BY inccount DESC LIMIT 10")
top10ResSQL: org.apache.spark.sql.DataFrame = [resolution: string, inccount: bigint]

scala> top10ResSQL.show
+--------------------+--------+
|          resolution|inccount|
+--------------------+--------+
|                NONE|  243538|
|       ARREST/BOOKED|   86766|
|        ARREST/CITED|   22925|
|   PSYCHOPATHIC_CASE|    8344|
|             LOCATED|    6878|
|           UNFOUNDED|    4551|
|COMPLAINANT_REFUS...|    4215|
|     JUVENILE_BOOKED|    2381|
|EXCEPTIONAL_CLEAR...|    1134|
|      JUVENILE_CITED|    1039|
+--------------------+--------+
```

SFPD 예 – Top 3 범죄 유형

- 가장 많은 3개의 범죄 유형(category)은?

```
// Top 3 categories
val top3Cat = sfpdDF.groupBy("category")
                    .count.sort($"count".desc)
top3Cat.show(3)
```

```
scala> val top3Cat = sfpdDF.groupBy("category").count.sort($"count".desc)
top3Cat: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [category: string, co
unt: bigint]

scala> top3Cat.show(3)
+--------------+-----+
|      category|count|
+--------------+-----+
| LARCENY/THEFT|96955|
|OTHER_OFFENSES|50611|
|  NON-CRIMINAL|50269|
+--------------+-----+
only showing top 3 rows
```


SFPD 예 – Top 3 범죄 유형, SQL 버전

- 가장 많은 3개의 범죄 유형(category)은?

```
// Top 3 categories : SQL
val top3CatSQL=spark.sql("SELECT category, count(incidentnum)
  AS inccount FROM sfpd GROUP BY category ORDER BY inccount
  DESC LIMIT 3")
top3CatSQL.show
```

```
scala> val top3CatSQL=spark.sql("SELECT category, count(incidentnum) AS inccount FROM s
fpd GROUP BY category ORDER BY inccount DESC LIMIT 3")
top3CatSQL: org.apache.spark.sql.DataFrame = [category: string, inccount: bigint]

scala> top3CatSQL.show
+--------------+--------+
|      category|inccount|
+--------------+--------+
| LARCENY/THEFT|   96955|
|OTHER_OFFENSES|   50611|
|  NON-CRIMINAL|   50269|
+--------------+--------+
```

SFPD 예 – Top 5 사건 해결 저장 (1)

- 가장 많은 10개의 사건 해결 유형의 결과 데이터프레임을 JSON 파일 형식으로 저장
  - write() 메서드로 출력
  - format() 메서드로 저장 양식 지정
  - save() 메서드로 저장
    - 인수로 저장되는 디렉토리 기술

// Save the top 10 resolutions to a JSON file.
top10ResSQL.write.format("json").save("/sparkdata/sfpd/output")

```
scala> top10ResSQL.write.format("json").save("/sparkdata/sfpd/output")

scala>
```


SFPD 예 – Top 5 사건 해결 저장 (2)

```
@MASTER:~$ hadoop fs -ls /sparkdata/sfpd/output
Found 2 items
-rw-r--r--   3 cssjlee supergroup          0 2017-07-31 14:55 /sparkdata/sfpd/output/_SUCCESS
-rw-r--r--   3 cssjlee supergroup        488 2017-07-31 14:55 /sparkdata/sfpd/output/part-00000
-7ce70990-6068-4489-b824-5aa32f84ed83.json
@MASTER:~$
@MASTER:~$ hadoop fs -cat /sparkdata/sfpd/output/part-00000-7ce70990-6068-4489-b824-5aa3
2f84ed83.json
{"resolution":"NONE","inccount":243538}
{"resolution":"ARREST/BOOKED","inccount":86766}
{"resolution":"ARREST/CITED","inccount":22925}
{"resolution":"PSYCHOPATHIC_CASE","inccount":8344}
{"resolution":"LOCATED","inccount":6878}
{"resolution":"UNFOUNDED","inccount":4551}
{"resolution":"COMPLAINANT_REFUSES_TO_PROSECUTE","inccount":4215}
{"resolution":"JUVENILE_BOOKED","inccount":2381}
{"resolution":"EXCEPTIONAL_CLEARANCE","inccount":1134}
{"resolution":"JUVENILE_CITED","inccount":1039}
@MASTER:~$
```

3. 사용자 정의 함수 생성

사용자 정의 함수

- 사용자 정의 함수 (User Defined Function, UDF)를 사용하여 맞춤형으로 적용
  - 데이터 프레임 연산에 적용
  - SQL 질의 에 적용
- 데이터프레임 연산에 적용
  - udf() 사용하여 인라인 생성

```
val func1 = udf((arguments) => { function definition })
```

- SQL 질의에 적용
  - SparkSession.udf.register() 사용하여 함수 등록

SQL 질의 사용자 정의 함수

- SQL 질의 함수 등록 방법
  - 함수 정의 후 등록
    - 등록 시 SQL에 적용되는 함수 이름, 정의된 함수가 인수로 전달

```
def funcname
.....
SparkSession.udf.register("sqlfuncname", funcname(_:type, ....) )
```

  - 인라인(inline)으로 함수 등록

```
SparkSession.udf.register("sqlfuncname", (arguments) => {
   function definition })
```


SFPD 예 – 연도 별로 사건 조사

- 연도 별로 사건 조사 예
- SFPD 데이터에서 날짜는 "dd/mm/yy "형식으로 저장
  - 연도 추출을 위해 마지막 '/ ' 다음의 문자열 추출
  - 추출된 년도를 기준으로 계산

SPDF 예 – 데이터프레임 연산 UDF (1)

- 연도 별로 사건 수 조회
  - UDF 함수 정의

```
// Define UDF for DataFrame operations to extract year
val getStr = udf((s: String) => {
val lastS = s.substring(s.lastIndexOf('/')+1)
lastS})
```

  - 데이터프레임 연산 적용

```
// Use UDF for incidents count by year
val yy =  sfpdDF.groupBy(getStr(sfpdDF("date"))).count
yy.show
```


SPDF 예 – 데이터프레임 연산 UDF (2)

```
scala> val getStr = udf((s: String) => {
     | val lastS = s.substring(s.lastIndexOf('/')+1)
     | lastS
     | })
getStr: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function1>
,StringType,Some(List(StringType)))

scala> val yy =  sfpdDF.groupBy(getStr(sfpdDF("date"))).count
yy: org.apache.spark.sql.DataFrame = [UDF(date): string, count: bigint]

scala> yy.show
+---------+------+
|UDF(date)| count|
+---------+------+
|       15| 80760|
|       13|152830|
|       14|150185|
+---------+------+
```

SFPD 예 – SQL 질의 UDF, 연도 별 사건 수 (1)

- 연도 별로 사건 수 조회
  - UDF 함수 정의

```
def get_year(s:String):String = {
val year = s.substring(s.lastIndexOf('/')+1)
year
}
```

  - UDF 함수 등록

```
spark.udf.register("getyear",get_year(_:String))
```

  - SQL 질의 적용

```
val incyearSQL=spark.sql("SELECT getyear(date),
   count(incidentnum) AS countbyyear FROM sfpd GROUP BY
   getyear(date) ORDER BY countbyyear DESC")
incyearSQL.collect.foreach(println)
```


SFPD 예 – SQL 질의 UDF, 연도 별 사건 수 (2)

```
scala> def get_year(s:String):String = {
     | val year = s.substring(s.lastIndexOf('/')+1)
     | year
     | }
get_year: (s: String)String

scala> spark.udf.register("getyear",get_year(_:String))
res9: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function1>,S
tringType,Some(List(StringType)))

scala> val incyearSQL=spark.sql("SELECT getyear(date), count(incidentnum) AS countbyyear FROM
sfpd GROUP BY getyear(date) ORDER BY countbyyear DESC")
incyearSQL: org.apache.spark.sql.DataFrame = [UDF(date): string, countbyyear: bigint]

scala> incyearSQL.collect.foreach(println)
[13,152830]
[14,150185]
[15,80760]
```

SFPD 예 – SQL 질의, 인라인 UDF 등록

- 인라인 함수 등록

```
// Inline UDF with SQL
spark.udf.register("getyear", (s: String) =>
   {s.substring(s.lastIndexOf('/')+ 1)})
```

- SQL 질의 적용

```
val incyearSQL=spark.sql("SELECT getyear(date),
   count(incidentnum) AS countbyyear FROM sfpd GROUP BY
   getyear(date) ORDER BY countbyyear DESC")
incyearSQL.collect.foreach(println)
```


SFPD 예 – SQL 질의 UDF, 2014년 사건 조사 (1)

- 2014년도의 사건의 범죄 유형, 해결 유형 및 범죄 발생 주소 조사

```
// Category, resolution and address of reported incidents in 2014
val inc2014 = spark.sql("SELECT category,address,resolution, date
   FROM sfpd WHERE getyear(date)='14'")
// Show 10 incidents in 2014
inc2014.collect.take(10)
// Show all incidents in 2014
inc2014.collect.foreach(println)
```

SFPD 예 – SQL 질의 UDF, 2014년 사건 조사 (2)

```
scala> val inc2014 = spark.sql("SELECT category,address,resolution, date FROM sfpd WHERE getye
ar(date)='14'")
inc2014: org.apache.spark.sql.DataFrame = [category: string, address: string ... 2 more fields
]

scala> inc2014.collect.take(10)
res15: Array[org.apache.spark.sql.Row] = Array([ASSAULT,0_Block_of_UNITEDNATIONS_PZ,NONE,12/31
/14], [NON-CRIMINAL,400_Block_of_POWELL_ST,NONE,12/31/14], [ASSAULT,700_Block_of_MONTGOMERY_ST
,NONE,12/31/14], [VANDALISM,FOLSOM_ST/SPEAR_ST,NONE,12/31/14], [NON-CRIMINAL,2800_Block_of_LEA
VENWORTH_ST,NONE,12/31/14], [NON-CRIMINAL,3300_Block_of_WASHINGTON_ST,NONE,12/31/14], [OTHER_O
FFENSES,3300_Block_of_WASHINGTON_ST,NONE,12/31/14], [LARCENY/THEFT,2000_Block_of_LAGUNA_ST,NON
E,12/31/14], [LARCENY/THEFT,CASTRO_ST/17TH_ST,NONE,12/31/14], [LARCENY/THEFT,300_Block_of_POWE
LL_ST,NONE,12/31/14])

scala> inc2014.collect.foreach(println)
```

```
[VANDALISM,800_Block_of_GREENWICH_ST,NONE,9/21/14]
[VEHICLE_THEFT,16TH_ST/DOLORES_ST,NONE,9/21/14]
[NON-CRIMINAL,PACIFIC_AV/SCOTT_ST,NONE,9/21/14]
[LARCENY/THEFT,DORE_ST/FOLSOM_ST,NONE,9/21/14]
[NON-CRIMINAL,700_Block_of_VALLEJO_ST,PSYCHOPATHIC_CASE,9/21/14]
[ASSAULT,0_Block_of_CORDOVA_ST,ARREST/CITED,9/21/14]
[DRUG/NARCOTIC,800_Block_of_MARKET_ST,ARREST/CITED,9/21/14]
[DRUNKENNESS,1000_Block_of_FITZGERALD_AV,NONE,9/21/14]
[NON-CRIMINAL,1000_Block_of_FITZGERALD_AV,NONE,9/21/14]
[WARRANTS,ELLIS_ST/MASON_ST,ARREST/BOOKED,9/21/14]
[ASSAULT,ELLIS_ST/MASON_ST,ARREST/BOOKED,9/21/14]
[SUSPICIOUS_OCC,2200_Block_of_PACIFIC_AV,NONE,9/21/14]
```


SFPD 예 – SQL 질의 UDF, 2015년 사건 조사 (1)

- 2015년도의 공공 기물 파손 범죄 유형의 해결 유형 및 범죄 발생 주소 조사

```
// Vandalism only in 2015 with address, resolution and category
val van2015 = spark.sql("SELECT category,address,resolution, date
   FROM sfpd WHERE getyear(date)='15' AND
   category='VANDALISM'")
van2015.count
van2015.collect.foreach(println)
```

SFPD 예 – SQL 질의 UDF, 2015년 사건 조사 (2)

```
scala> val van2015 = spark.sql("SELECT category,address,resolution, date FROM sfpd WHERE getye
ar(date)='15' AND category='VANDALISM'")
van2015: org.apache.spark.sql.DataFrame = [category: string, address: string ... 2 more fields
]

scala> van2015.count
res17: Long = 3898

scala> van2015.collect.foreach(println)
```

```
[VANDALISM,300_Block_of_9TH_AV,NONE,1/1/15]
[VANDALISM,MOSCOW_ST/ITALY_AV,NONE,1/1/15]
[VANDALISM,FULTON_ST/28TH_AV,NONE,1/1/15]
[VANDALISM,5TH_ST/MINNA_ST,NONE,1/1/15]
[VANDALISM,COLE_ST/PARNASSUS_AV,NONE,1/1/15]
[VANDALISM,26TH_ST/CAPP_ST,ARREST/BOOKED,1/1/15]
[VANDALISM,0_Block_of_MARINA_BL,ARREST/BOOKED,1/1/15]
[VANDALISM,700_Block_of_HEAD_ST,NONE,1/1/15]
[VANDALISM,100_Block_of_LENOX_WY,NONE,1/1/15]
[VANDALISM,200_Block_of_LEAVENWORTH_ST,NONE,1/1/15]
[VANDALISM,DIVISION_ST/KING_ST,ARREST/BOOKED,1/1/15]
[VANDALISM,1900_Block_of_42ND_AV,NONE,1/1/15]
[VANDALISM,100_Block_of_BROADWAY_ST,NONE,1/1/15]
[VANDALISM,1200_Block_of_CONNECTICUT_ST,NONE,1/1/15]
[VANDALISM,ELLIS_ST/STEINER_ST,NONE,1/1/15]
[VANDALISM,600_Block_of_COLUMBUS_AV,NONE,1/1/15]
[VANDALISM,0_Block_of_BRENTWOOD_AV,NONE,1/1/15]
[VANDALISM,1900_Block_of_PALOU_AV,NONE,1/1/15]
```


과제

- 자신만의 데이터를 사용하여 앞에서 배운 스파크를 적용하고 실행
  - 텀 프로젝트와 관련하여 연관성이 있는 데이터

참고 자료

- MapR Academy DEV 361 Build and Monitor ApacheSpark Applications
  - http://learn.mapr.com/dev-360-apache-spark-essentials
  - Lesson 5: Work with DataFrames
- Spark SQL, DataFrames and Datasets Guide
  - https://spark.apache.org/docs/latest/sql-programming-guide.html#dataframes
- Converting Spark RDD to DataFrame and Dataset. Expert Opinion.
  - https://medium.com/@InDataLabs/converting-spark-rdd-to-dataframe-and-dataset-expert-opinion-826db069eb5
- M. Armbrust, R. Xin, C. Lian, Y. Huai, D. Liu, J. Bradley, X. Meng, T. Kaftan, M. Franklin, A. Ghodsi and M. Zaharia.
  Spark SQL: Relational Data Processing in Spark. *SIGMOD 2015*.