스파크로
구축하는 분산처리
빅데이터 플랫폼

1.Spark 기초와 빅데이터 플랫폼의 개념 및 설계

1.1 스파크 SQL

1. 스파크 DataFrame

스파크 DataFrame

- 정형 데이터는 데이터 구조를 미리 파악할 수 있다는 장점 덕분에 여러 빅데이터 사례에서도 자 주 활용
- DataFrame은 이러한 정형 데이터를 효과적으로 다룰 수 있는 방법을 제공
- DataFrame을 사용하면 칼럼 이름으로 데이터를 참조할 수 있음
- 데이터를 조작할 수 있는 SQL 쿼리를 이용해 데이터에 접근할 수 있음
- DataFrame은 다 양한 소스의 데이터를 손쉽게 통합할 수 있도록 지원함
- 스파크 SQL에서는 DataFrame을 테이블 카탈로그의 테이블로 등록

1. 스파크 DataFrame

스파크 DataFrame

- 테이블 카탈로 그는 데이터 자체를 저장하지 않는 대신 정형 데이터로 접근하는 방법만 저장
- DataFrame을 테이블 카탈로그에 등록하면 다른 스파크 애플리케이션에서도 DataFrame 이름을 이용해 이 데이터에 질의를 수행
- 외부의 써드-파티 애플리케이션에서도 표준 JDBC 및 ODBC 프로토콜로 스파크에 접속한 후, 등록된 DataFrame 테이블의 데이터 에 SQL 쿼리를 수행할 수 있음

1. 스파크 DataFrame

스파크 DataFrame

- 스파크 쓰리프트 서버는 원격 쿼리 기능을 지원 하는 스파크 컴포넌트로, JDBC 클라이언트가 요청한 쿼리를 DataFrame API를 이용해 스파크 잡 형태로 실행 함
- DataFrame DSL을 사용해 두 테이블을 조인하는 스파크 애플리케이션
- JDBC를 사용해 스파크 쓰리프트 서버로 연결하고 SQL 쿼리를 실행하는 비-스파크 애플리케이션
- 테이블 카탈로그는 스파크 컨텍스트를 종료해도 계속 유지되는 영구 적인 (permanent) 메타데이터 저장소

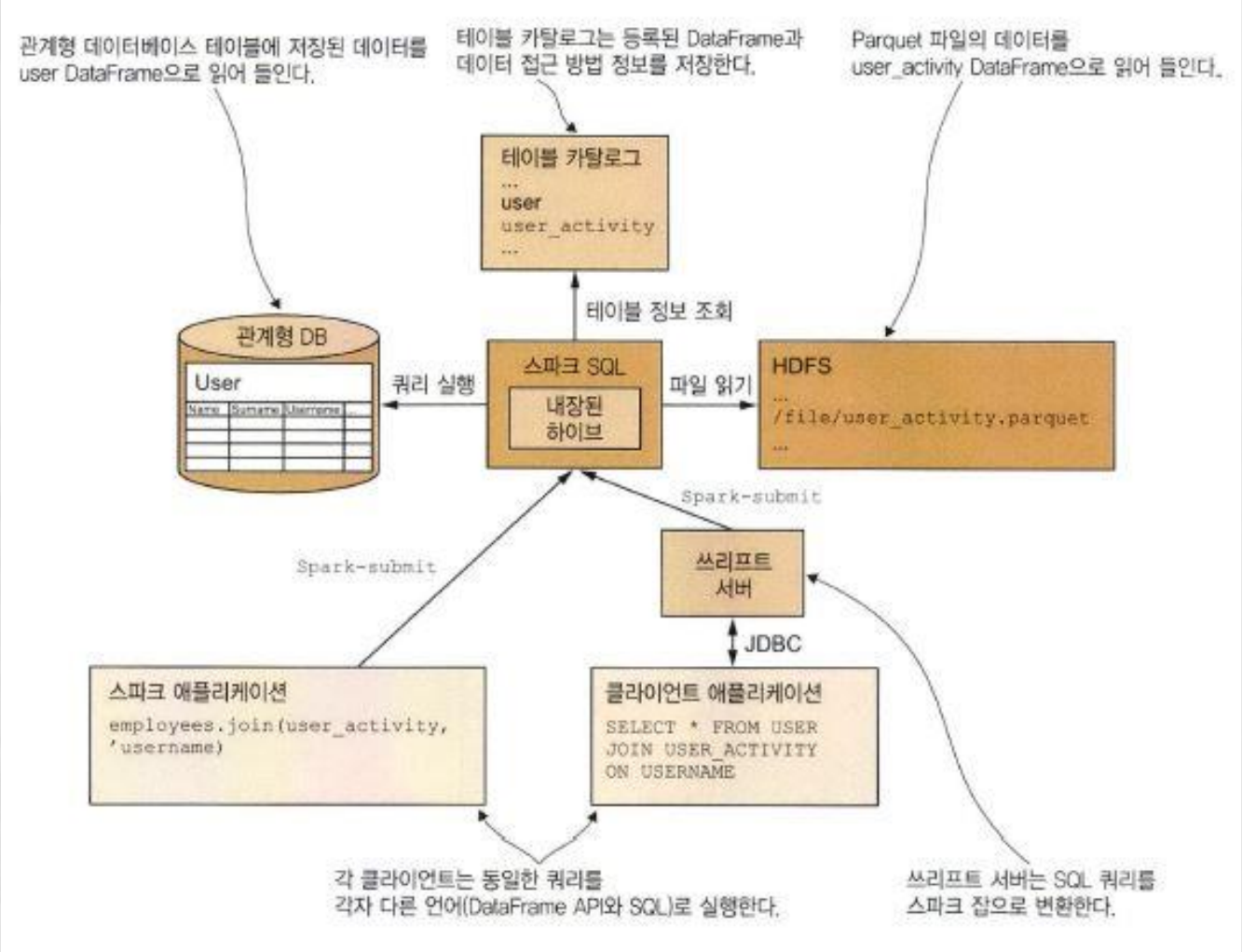
1. 스파크 DataFrame

스파크 DataFrame

- 하이브(Hive) 지원 기능이 포함된 스파크에서만 사용
- 하이브는 하둡의 맵리듀스를 한 단계 더 추상화한 분산 웨어 하우스 시스템
- 하이브는 SQL과 유사한 HiveQL이라는 독자적 인 쿼리 언어를 사용
- HiveQL 쿼리 작업은 맵리듀스 잡뿐만 아니라 스파크 잡으로도 실행
- 하이브 지원 기능이 포함된 스파크는 하이브에 필요한 모든 의존 라이브러리를 포함

1. 스파크 DataFrame

스파크 DataFrame



1. 스파크 DataFrame

스파크 DataFrame

- DataFrame에는 모든 칼럼의 타입을 미 리 지정하므로 RDD에 비해 더 간결하고 손쉽게 쿼리를 작성할 수 있음
- 기존 RDD를 변환하는 방법
- SQL 쿼리를 실행하는 방법
- 외부 데이터에서 로드 하는 방법

1. 스파크 DataFrame

RDD에서 DataFrme 생성

- DataFrame을 생성할 때는 데이터를 먼저 RDD로 로드한 후 DataFrame으로 변환하는 방법
- 비정형 데이터에 DataFrame API 를 사용하려면 먼저 이 데이터를 RDD로 로드하고 변환한 후 이 RDD에서 DataFrame을 생성 해야 함
- RDD에서 DataFrame을 생성하는 방법
  - 로우의 데이터를 튜플 형태로 저장한 RDD를 사용하는 방법
  - 케이스클래스를 사용하는 방법
  - 스키마를 명시적으로 지정하는 방법

1. 스파크 DataFrme

스파크 DataFrame

- 스파크 DataFrame과 SQL 표현식을 사용하려면 먼저 SparkSession 객체 준비
- 스파크 셸은 SparkSession을 spark라는 이름의 변수로 제공
- scala> import org.apache.spark.sql.SparkSession
- scala> val spark = SparkSession.builder().getOrCreate()
- SparkSession은 SparkContext와 SQLContext를 통합한 래퍼(wrapper) 클래스
- 스파크는 RDD 를 Data Fra me 으로 자동 변환하는 데 필요한 암시적 스칼라 메서드들을 제공
- spark라는 변수 이름으로 SparkSession 객체를 생성

1. 스파크 DataFrme

스파크 DataFrame

- scala> import spark.implicits._
- SparkSession의 implicits 객체는 Dataset Encoder가 정의된 객체로 구성한 RDD에 toDF라는 메서드를 추가
- Encoder는 스파크 SQL이 내부적으로 JVM 객체를 테이블 형식으로 변환하는 데 사용하는 트레잇

1. 스파크 DataFrme

데이터셋 로드

- 원본 데이터는 XML 형식이지만, 파일을 CSV 형식으로 변환한 데이터를 사용
  - commentCount : 이 포스트 에 달린 댓글 개수
  - lastActivityDate : 마지막 수정 날짜 및 시각
  - ownerIlserId : 포스트를 게시한 사용자 ID
  - body : 질문 및 답변 내용
  - score : '좋아요'와 '싫어요' 개수로 계산한 총 점수
  - creationDate : 생성 날짜 및 시각
  - viewCount: 조회 수
  - title : 질문 제목
  - tags : 질문에 달린 태그들
  - answerCount : 질문에 달린 답변 개수
  - acceptedAnswerId: 답변이 채택된 경우 채택된 답변의 ID
  - postTypeId: 이 포스트의 유형. 1: 질문 , 2: 답변
  - id: 이 포스트의 고유 ID

1. 스파크 DataFrme

데이터셋 로드

- scala> val ltPostsRows = sc.textFile("italianPosts.csv")
- scala> val itPostsSplit = itPostsRows.map(x => x.split("~"))
- 문자열 배열로 구성된 RDD를 반환
- RDD의 toDF 메서드를 호출하면 문자열 배열 타입의 단일 칼럼을 가진 Data Frame 을 얻을 수 있음

1. 스파크 DataFrme

튜플 형식의 RDD에서 DataFrame 생성

- RDD의 배열을 튜플로 변환하고 대아를 호출해서 DataFrame을 생성
- scala> val itPostsRDD = itPostsSplit.map(x =>

  (x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8),x(9),x(10),x(11),x(12)))
- scala> val itPostsDFrame = itPostsRDD.toDF()
- scala> itPostsDFrame.show(10)
- val itPostsDF = itPostsRDD.toDF("commentCount", "lastActivityDate",

  "ownerUserId", "body", "score", "creationDate", "viewCount", "title", "tags",

  "answerCount", "acceptedAnswerId", "postTypeId", "id")

1. 스파크 DataFrme

튜플 형식의 RDD에서 DataFrame 생성

- scala> itPostsDF.printSchema
- printSchema 메서드는 DataFrame이 가진 칼럼 정보를 출력
- 모든 칼럼의 타입이 String이며 nullable(NULL 값을 허용하는 칼럼)인 것
- 칼럼 중에서 Id로 끝나는 칼럼들은 모두 long 타입이고, count와 score 관련 칼럼들은 정수형 타입이고, 날짜를 나타내는 칼럼들은 모두 타임스탬프로 지정해야 함.

1. 스파크 DataFrme

케이스 클래스를 사용해 RDD를 DataFrame으로 변환

- RDD를 DataFrame으로 변환하는 두 번째 방법은 RDD의 각 로우를 케이스 (case) 클래스로 매핑 한 후 toDF 메서드를 호출하는 것
- 데이터셋의 각 로우를 저장할 Post 클래스를 선언
- import java.sql.Timestamp
- Post 클래스에서는 nullable 필드를 Option[T] 타입으로 선언
- StringImplicits 객체 내에 암시적 클래스로 선언된 StringImprovements는 세 가지 Safe 메서드 를 스칼라의 String 클래스에 암시적으로 추가함

1. 스파크 DataFrme

케이스 클래스를 사용해 RDD를 DataFrame으로 변환

- catching 함수는 scala.util.control.Exception.Catch 타입의 객체를 반환하는데, 이 객체의 opt 메서드는 사용자가 지정한 함수(예: s.toInt) 결과를 Option 객체로 매핑
- 암시적 클래스를 임포트한 후 이 클래스의 메서드를 사용해 각 필드(문자열)를 적절한 타입의 객체로 파싱
- scala>itPostsDFCase.printSchema

1. 스파크 DataFrme

스키마를 지정해 RDD를 DataFrame으로 변환

- SparkSession의 createDataFrame 메서드를 사용 하는 것
- Row 타입의 객체를 포함하는 RDD와 StnjctType 객체를 인자로 전달해 호출 할 수 있음
- StructType은 스파크 SQL의 테이블 스키마를 표현하는 클래스
- StructType 객 체는 테이블 칼럼을 표현하는 StructField 객체를 한 개 또는 여러 개 가질 수 있음

1. 스파크 DataFrme

스키마 정보 가져오기

- DataFrame의 schema 필드를 사용해 StructType 객체로 정의된 스키마를 참조 할 수 있음
- columns와 dtypes 메서드로 DataFrame의 스키마 정보를 확인할 수 있음
- colomins 메서 드는 칼럼 이름 목록을 반환하며, dtypes 메서드는 각 칼럼 이름과 타입으로 구성된 튜플 목록을 반환

```
scala> itPostsDFCase.columns
res0: Array[String] = Array(commentCount, lastActivityDate, ownerUserId,
   body, score, creationDate, viewCount, title, tags, answerCount,
   acceptedAnswerId, postTypeId, id)
scala> itPostsDFStruct.dtypes
res1: Array[(String, String)] = Array((commentCount,IntegerType),
   (lastActivityDate,TimestampType), (ownerUserId,LongType),
```

1. 스파크 DataFrme

기본 DataFrame API

- DataFrame의 DSL은 스파크 SQL의 핵심 개념으로 DataFrame 데이터를 다룰 수 있는 다양한 기능을 제공
- DataFrame의 DSL은 관계형 데이터베이스의 일반 SQL 함수와 유사한 기능을 제공
- DataFrame의 데이터를 직접 변경하지 못하며, 반드시 새로운 DataFrame으로 변환

1. 스파크 DataFrme

기본 DataFrame API –컬럼 선택

- 대부분의 DataFrame DSL 함수는 Column 객체를 입력받을 수 있음
- scala> val postsDf = itPostsDFStuct
- scala> val postsIdBody = postsDf.select("id“, "body")
- val postsIdBody = postsDf.select(postsDf.col("id"), postsDf.col("body"))
- val postsIdBody = postsDf.select(Symbol("id”), Symbol(“body”))
- val postsIdBody = postsDf.select(’id, 'body)

1. 스파크 DataFrme

기본 DataFrame API – 데이터 필터링

- DataFrame의 데이터 필터링 함수에는 where와 filter
- where 와 filter 함수는 Column 객체 또는 문자열 표현식을 인수로 받음
- scala> postsIdBody.filter('body contains "Italian").count
- scala> val noAnswer = postsDf.filter((1postTypeId === 1) and

  (’acceptedAnswerId isNull))
- scala> val firstTenQs = postsDf .filter (’postTypeId === l).limit(10)

1. 스파크 DataFrame

기본 DataFrame API – 컬럼을 추가하거나 컬럼 이름 변경

- 칼럼 이 름은 withColumnRenamed 함수로 변경
- val firstTenQsRn = firstTenQs.withColumnRenamed(ownerUserId", "owner")
- withColumn 함수에 칼럼 이름과 Column 표현식을 전달해 DataFrame에 신규 칼럼을 추가할 수 있음

1. 스파크 DataFrme

기본 DataFrame API – 데이터 정렬

- DataFrame의 orderBy와 sort 함수는 한 개 이상의 칼럼 이름 또는 Column 표현식을 받고 이를 기준으로 데이터를 정렬 함
- Column 클래스에 asc나 desc 연산자를 사용해 정렬 순서를 지정할 수 있음
- 정렬 순서를 지정하지 않으면 오름차순이 기본으로 적용 됨

1. 스파크 DataFrame

SQL 함수로 데이터에 연산 수행

- 스파크의 SQL 함수
  - 스칼라 함수: 각 로우의 단일 칼럼 또는 여 러 칼럼 값을 계산해 단일 값을 반환하는 함수
  - 집계 함수: 로우의 그룹에서 단일 값을 계산하는 함수
  - 윈도함수: 로우의 그룹에서 여러 결과값을 계산하는 함수
  - 사용자 정의 함수: 커스텀 스칼라 함수 또는 커스텀 집계 함수

1. 스파크 DataFrame

SQL 함수로 데이터에 연산 수행

- 내장 스칼라 함수와 내장 집계 함수
  - 스칼라(scalar) SQL 함수는 각 로우의 단일 칼럼 값이나 여러 칼럼 값을 계산해 해당 로우의 단일 결과 값으로 반환 함
  - 절댓값을 계산하는 abs나 지수 결과를 계산하는 exp, 문자열 일 부를 추출하는 substring 등이 스칼라 함수에 해당
  - 집계 함수는 로우 그룹에서 단일 값을 반환
  - 로우 그룹의 최솟값을 계산하는 min이나 평균값을 계산하는 avg 함수 등이 집계 SQL 함수에 해당

1. 스파크 DataFrame

SQL 함수로 데이터에 연산 수행

- 스파크가 제공하는 스칼라 함수
  - 수학 계산: abs(절댓값을 계산), hypot(칼럼 값이나 스칼라 값 두 개를 각각 삼각형의 밑변과 높이로 가정하고, 이 삼각형의 빗변 길이를 계산), log(로그 값을 계산), cbrt(세제곱근(cube root)을 계산)
  - 문자열 연산: length(문자열 길이를 반환) , trim(문자열 왼쪽과 오른쪽에 있는 불필요한 공백을 잘라 냄) , concat(복수의 입력 문자열을 이어 붙임)
  - 날짜 및 시간 연산: year(날짜 칼럼의 연도를 반환), date_add(날짜 칼럼에 입력된 일수 만큼 더함), datediff함수(두 날짜 간의 차이(일 단위 차이)를 계산)

1. 스파크 DataFrame

SQL 함수로 데이터에 연산 수행

- 윈도 함수

함수	설명
first (column)	프레임에 포함된 로우 중에서 첫 번째 로우 값을 반환한다.
last (column)	프레임에 포함된 로우 중에서 마지막 로우 값을 반환한다.
lag (column, offset, [default])	프레임에 포함된 로우 중에서 현재 로우를 기준으로 offset만큼 뒤에 있는 로우 값을 반환한다. 이 위치에 로우가 없다면 주어진 default 값을 그대로 사용한다.
lead (column, offset, [default])	프레임에 포함된 로우 중에서 현재 로우를 기준으로 offset만큼 앞에 있는 로우 값을 반환한다. 이 위치에 로우가 없다면 주어진 default 값을 그대로 사용한다.
ntile (n)	프레임에 포함된 로우를 그룹 n개로 분할하고, 현재 로우가 속한 그룹의 인덱스를 반환한다. 프레임에 포함된 로우 개수가 n으로 나누어 떨어지지 않고 몫이 임의의 정수 x와 x+1 사이의 실수일 때, 각 그룹은 로우 x개 또는 x+1개를 각각 나누어 가진다. 이때 순서가 빠른 그룹에 로우 x+1개를 먼저 할당한다.
cume_dist	프레임에 포함된 로우 중에서 현재 처리하고 있는 로우의 값보다 작거나 같은 다른 로우들이 프레임에서 차지하는 비율을 반환한다.
rank	프레임에 포함된 로우 중에서 현재 처리하고 있는 로우 순위를 반환한다(예: 1위, 2위 등). 로우 순위는 특정 칼럼 값을 기준으로 계산한다.
dense_rank	프레임에 포함된 로우 중에서 현재 처리하고 있는 로우 순위를 반환한다(예: 1위, 2위 등). 이 함수가 rank 함수와 다른 점은 값이 동일한 로우에 동일한 순위를 부여하고, 다음 순위로 바로 이어진다는 점이다. 예를 들어 로우 세 개가 동일하게 1위 값을 가질 경우, 세 로우 모두에 1위를 부여하고 그다음 네 번째 로우에는 2위를 부여한다.[9]

1. 스파크 DataFrame

SQL 함수로 데이터에 연산 수행

- 윈도 함수

함수	설명
percent_rank	프레임에 포함된 로우 중에서 현재 처리하고 있는 로우 순위를 프레임의 전체 로우 개수로 나누어 반환한다.
row_number	프레임에 포함된 로우 중에서 현재 처리하고 있는 로우 순번(sequential number)을 반환한다.

- org.apache.spark.sql.expressions.Window 클래스의 정적(static) 메서드
  - partitionBy 메서드를 사용해 단일 칼럼 또는 복수 칼럼을 파티션의 분할 기준으로 지정(집계 함수에 사용하는 groupBy의 분할 기준과 같은 원리를 적용)
  - orderBy 메서드로 파티션의 로우를 정렬할 기준을 지정하는 것(데이터셋 전체를 대상)
  - partitionBy와 orderBy를 모두 사용하는 것

1. 스파크 DataFrame

결측 값 다루기

- 데이터 값이 null이 거나 아예 비어 있거나, 또는 결측 값에 준하는 문자열 상수(예: N/A 또는 unknown)를 데이터에 포함 함
- DataFrame의 na 필드로 제공되는 DataFrameNaFunctions를 활용해 결측 값이 있는 데이터 인스턴스를 적절하게 처리할 수 있음
- 보통은 null 또는 NaN(Not a Number를 뜻하는 스칼라 상수) 등 결측 값을 가진 로우를 DataFrame에서 제외하거나 결측 값 대신 다른 상수를 채워 넣거나 결측 값에 준하는 특정 칼럼 값을 다른 상수로 치환하는 방법을 사용

1. 스파크 DataFrame

결측 값 다루기

- drop 메서드를 인수 없이 호출하면 최소 칼럼 하나 이상에 null이나 NaN 값을 가진 모든 로우를 DataFrame에서 제외할 수 있음
- fill 함수를 사용해 nuU 또는 NaN 값을 double 상수 또는 문자열 상수로 채울 수 있으며, fill 함수의 인수를 하나만 지정하면 이 인수 값을 모든 칼럼의 결측 값을 대체할 상수로 사용
- replace 함수를 사용해 특정 칼럼의 특정 값을 다른 값으로 치환

1. 스파크 DataFrame

DataFrame을 RDD로 변환

- 결과로 반환된 RDD는 org.apache.spark.sql.Row 타입의 요소로 구성
- Row 클래스는 칼럼 번호로 해당 칼럼 값을 가져올 수 있는 다양한 get* 함수 [getStringIndex) , getInt(index), getMap(index) 등]를 제공
- DataFrame 데이터와 파티션을 map이나 flatMap , mapPartitions 변환 연산자 등으로 매핑하면 실제 매핑 작업은 하부 RDD에서 실행 됨
- DataFrame의 변환 연산자는 DataFrame 스키마(즉, RDD 스키마)를 변경할 수 있음 즉 칼럼 순서나 개수, 타입을 변경할 수 있음

1. 스파크 DataFrame

데이터 그룹핑

- DataFrame API는 매우 직관적인 데이터 그루핑 기능을 제공
- DataFrame의 데이터 그루핑은 groupBy 함수로 시작
- 칼럼 이름 또는 Column 객체의 목록을 받고 GroupedData 객체를 반환
- GroupedData는 groupBy에 지정한 칼럼들의 값이 모두 동일한 로우 그룹들을 표현한 객체로, 각 그룹을 대상으로 값을 집계할 수 있는 표준 집계 함수 kount ， sum, max, min, avg)를 제공 함
- GroupedData의 각 집계 함수는 groupBy에 지정한 칼럼들과 집계 결과를 저장한 추가 칼럼으로 구성된 Data Frame 을 반환함

1. 스파크 DataFrame

데이터 그룹핑

- agg 함수를 사용해 서로 다른 칼럼의 여러 집계 연산을 한꺼번에 수행할 수 있음
- agg 함수에는 org.apache.spark.sql.functions 집계 함수를 사용한 칼럼 표현식을 한 개 또는 여러 개 전달하거나, 각 칼럼 이름과 집계 함수 이름을 매핑한 Map 객체를 전달할 수 있음

```
scala> postsDfNew.groupBy('ownerUserId).
  agg(max('lastActivityDate), max('score)).show(10)
scala> postsDfNew.groupBy('ownerUserId).
  agg(Map("lastActivityDate" -> "max", "score" -> "max")).show(10)
```

1. 스파크 DataFrame

데이터 그룹핑 - 사용자 정의 집계 함수

- DataFrame의 rollup과 cube 함수는 추가적인 데이터 그루핑 및 집계 기능을 제공
- rollup과 cube는 지정된 칼럼의 부분 집합을 추가로 사용해 집계 연산을 수행
- cube는 칼럼의 모든 조합(combination)을 대상으로 계산하는 반면, rollup은 지정된 칼럼 순서를 고려한 순열(pemutation)을 사용
- rollup과 cube 함수는 groupBy와 마찬가지로 DataFrame 클래스로 사용할 수 있음

1. 스파크 DataFrame

데이터 조인

- 서로 다른 두 DataFrame에 저장한 연관 데이터를 조인해 값이 공통된 로우들로 새로운 DataFrame 을 만들어야 하는 경우
- 두 DataFrame을 조인하려면 한 DataFrame의 join 함수에 다른 하나의 DataFrame을 전달하고, 하나 이상의 칼럼 이름 또는 Column 정의를 조인 기준으로 지정
- 칼럼 이름을 조인 기준으로 사용할 때는 해당 칼럼이 양쪽에 모두 있어야 함

2. 스파크 SQL 명령

테이블 카탈로그와 하이브 메타스토어

- 대부분의 SQL 연산자는 테이블과 칼럼 이름을 바탕으로 SQL 연산을 수행
- 스파크는 사용자가 등록한 테이블 정보를 테이블 카탈로그(table catalog)에 저장
- 하이브를 지원하는 SparkSession에서는 테이블 카탈로그가 하이브 메타스토어 (metastore)를 기반으로 구현
- 하이브 메타스토어는 영구적인 데이터베이스로, 스파크 세션 을 종료하고 새로 시작해도 DataFrame 정보를 유지함
- createOrReplaceTempView 메소드

2. 스파크 SQL 명령

SQL 쿼리 실행

- SQL 표현식은 SparkSession의 sql 함수로 실행
- 하이브 지원 SparkSession 을 사용하면 하이브가 제공하는 명령 및 데이터 타입을 대부분 활용 할 수 있음
- 스파크 셸에서는 sql 함수를 지동으 로 임포트하므로(import spark,sql) SQL 명령을 바로 실행할 수 있음
- 스파크 SQL로 ALTER TABLE이나 DROP TABLE 등 DDL(Data Definition Language)도 사용할 수 있음

2. 스파크 SQL 명령

SQL 쿼리 실행

- spark-sql의 -e 인수를 사용해 스파크 SQL 셸을 들어가지 않고도 SQL 쿼리를 실행할 수 있음
- $ spark-sql -e "select substring(title,0, 70) from posts where postTypeId= 1 order by creationDate desc limit 3“
- spark-sql의 -f 인수는 파일에 저장된 SQL 명령을 실행
- -i 인수로 초기화 SQL 파일을 지정할 수 있음

2. 스파크 SQL 명령

쓰리프트 서버로 스파크 SQL 접속

- 스파크 쓰리프트라는 JDBC(또는 ODBC) 서버를 이용해 원격지에서 SQL 명령을 실행할 수 있음
- JDBC(또는 ODBC)는 관계형 데이터베이스의 표준 접속 프로토콜이므로 쓰리프트 서버를 이용 해 관계형 데이터 베이스와 통신할 수 있는 모든 애플리케이션에서 스파크를 사용할 수 있음
- 쓰리프트 서버는 여러 사용자의 JDBC 및 ODBC 접속을 받아 사용자의 쿼리를 스파크 SQL 세션으로 실행하는 독특한 스파크 애플리케이션임

2. 스파크 SQL 명령

쓰리프트 서버로 스파크 SQL 접속

- 쓰리프트 서버로 전달된 SQL 쿼리는 DataFrame 으로 변환한 후 최종적으로 RDD 연산으로 변환해 실행하며, 실행 결과는 다시 JDBC 프로토콜로 반환
- 쿼리가 참조하는 DataFrame은 쓰리프트 서버가 사용하는 하 이브 메타스토어에 미리 영구적으로 등록
- 리프트 서버는 스파크의 sbin 디렉터리 아래 start-thriftserver.sh 명령으로 시작
- 원격 메타스토어 데이터베이스를 이용하면 --jars 인수로 데이터베이스 접속에 사용할 JDBC 드라이버의 JAR 파일 위치를 쓰리프트 서버에 전달

2. 스파크 SQL 명령

쓰리프트 서버로 스파크 SQL 접속

- 쓰리프트 서버는 10000번 포트를 기본으로 사용
- 쓰리프트 서버의 포트 번호와 호스트네임을 변경하려면 HIVE_SERVER2_THRIFT_PORT와 HIVE_SERVER2_THRIFT_BIND_HOST 환경 변수를 설정하거나 하이브 환경 변수(hive.server2.thrift.port와 hive.server2.thrift.bind.host)를 start-thriftserver.sh의 --hiveconf 매개변수로 전달
- 스파크의 bin 디렉터리에 있는 비라인(Beeline)은 쓰리프트 서버로 접속할 수 있는 하이브의 명령

```
$ beeline -u jdbc:hive2://<server_name>:<port> -n <username> -p <password>
Connecting to jdbc:hive2://<server_name>:<port>
Connected to: Spark SQL (version 1.5.0)
Driver: Spark Project Core (version 1.5.0)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 1.5.0 by Apache Hive
0: jdbc:hive2://<server_name>:<port>>
```

2. DataFrame을 저장하고 불러오기

기본 데이터 소스

- 스파크는 다양한 기본 파일 포맷 및 데이터베이스를 지원
- JDBC와 하이브를 비롯해 JSON, ORC, Parquet 파일 포맷 등이 해당
- MySQL 및 PostgreSQL 관계형 데이터베이스와 스파크를 연동하는 Dialect 클래스(JDBC의 JdbcType 클래스와 스파크 SQL의 DataType 클래스를 매핑하는 클래스)를 제공
- 기본으로 지원하는 데이터 포맷에는 JSON, ORC, Parquet가 있음

2. DataFrame을 저장하고 불러오기

기본 데이터 소스

- 스파크는 JSON 스키마를 자동으로 유추할 수 있기 때문에 외부 시스템과 데이터를 주고받는 포맷 으로 매우 적합
- JSON의 단점은 데이터를 영구적으로 저장하는 데 사용하기에는 저장 효율이 떨어진다는 단점
- JSON의 장점은 포맷이 간단하고 사용이 간편하며 사람이 읽을 수 있는 형태로 데이터를 저장한다는 것

2. DataFrame을 저장하고 불러오기

기본 데이터 소스

- ORC(Optimized Row Columnar)는 하둡의 표준 데이터 저장 포맷인 RCFile보다 효율적으로 하이브의 데이터를 저장하도록 설계된 파일 포맷
- ORC는 칼럼형 포맷
- 칼럼형 포맷은 각 로우의 데이터를 순차적으로 저장하는 로우 포맷과 달리 각 칼럼의 데이터를 물리적으로 가까운 위치에 저장하는 방식을 의미
- ORC는 로우 데 이터를 스트라이프(stripe) 여러 개로 묶어서 저장하며, 파일 끝에는 파일 푸터(File footu)와 포스트 스크립트(postscripl) 영역이 있음

2. DataFrame을 저장하고 불러오기

기본 데이터 소스

- Parquet는 불필요한 의존 관계를 가지지 않으며 특정 프레임워크에 종속되지 않도록 설계
- 독립성 덕분에 하둡 생태계에서는 ORC 포맷보다 Parquet 포맷을 더 널리 사용
- 칼럼별로 압축 방식을 지정할 수 있다는 점에서 ORC와 다름
- Parquet는 중첩된(nested) 복합 데이터 구조에 중점을 두고 설계되어서 ORC 파일 포맷보다 이러한 중첩 구조의 데이터셋을 더 효율적으로 다룰 수 있음

2. DataFrame을 저장하고 불러오기

데이터 저장

- DataFrame의 데이터는 write 필드로 제공되는 DataFrameWriter 객체를 사용해 저장
- saveAsTable 메서드 외에도 save나 insertInto 메서드로 데이터를 저장
- saveAsTable과 insertInto 메서드는 데이터를 하이브 테이블로 저장하며 이 과정에서 메타스토어를 활용
- save 메서드는 데이터를 파일에 저장
- 하이브 미지원 SparkSession을 사용하면 saveAsTable과 insertInto 메서드는 하이브 테이블 대신 임시 테이블에 데이터를 저장

2. DataFrame을 저장하고 불러오기

데이터 저장

- Writer 설정 (DataFrameWriter)
- saveAsTable 메서드로 데이터 저장
- insertInto 메서드로 데이터 저장
- save 메서드로 데이터 저장
- 단축 메서드로 데이터 저장
- jdbc 메서드로 관계형 데이터베이스에 데이터 저장

2. DataFrame을 저장하고 불러오기

데이터 불러오기

- SparkSession의 read 필드로 제공되는 org.apache.spark.sql.DataFrameReader 객체를 사용해 데이터를 Data Frame 으로 불러 옴
- DataFrameReader 설정 함수로 format, option, options를 사용할 수 있음
- DataFrameReader의 load 메서드를 사용해 데이터 소스에서 데이터를 불러 옴
- table 함수를 사용해 하이브 메타스토어에 등록된 테이블에서 DataFrame을 불러올 수 있음

```
val postsDf = spark.read.table("posts")
val postsDf = spark.table("posts")
```

2. DataFrame을 저장하고 불러오기

데이터 불러오기

- jdbc 메서드로 관계형 데이터베이스에서 데이터 불러오기

```
scala> sql("CREATE TEMPORARY TABLE postsjdbc "+
"USING org.apache.spark.sql.jdbc "+
"OPTIONS ("+
"url 'jdbc:postgresql://postgresrv/mydb',"+
"dbtable 'posts',"+
"user 'user',"+
"password 'password')")
scala> val result = sql("select * from postsjdbc")
```

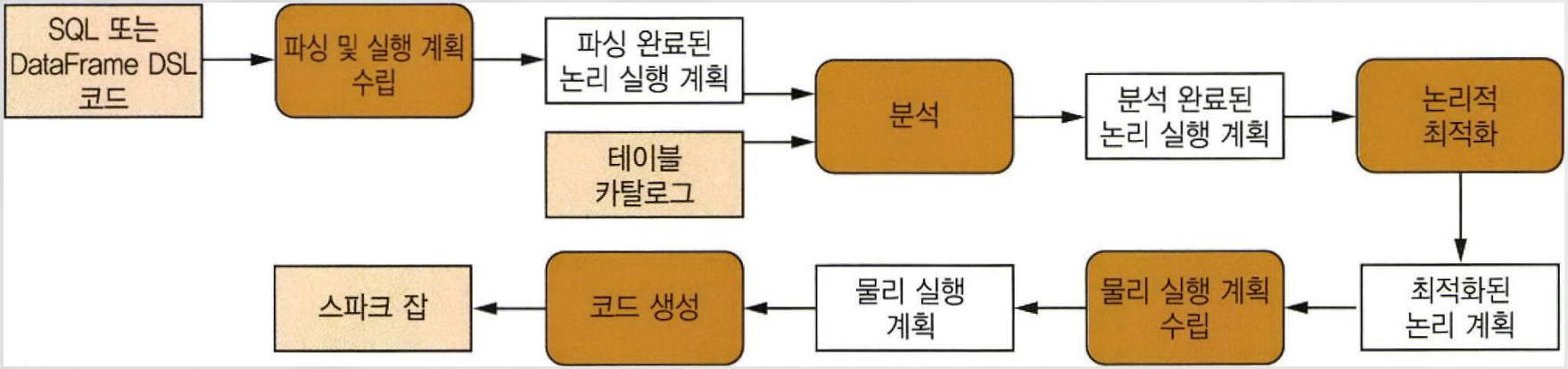
- sql 메서드로 등록한 데이터 소스에서 데이터 불러오기

```
scala> sql("CREATE TEMPORARY TABLE postsParquet "+
  "USING org.apache.spark.sql.parquet "+
  "OPTIONS (path '/path/to/parquet_file')")
scala> val resParq = sql("select * from postsParquet")
```

3. 카탈리스트 최적화 엔진

카탈리스트 엔진

- 카탈리스트 엔진이 SQL 및 DSL 표현식을 RDD 연산으로 변환하는 과정(파싱 및 분석. 논리적 최적화, 물리 실행 계획 수립, 코드 생성 단계로 진행



Any Questions?