Outline

Module 1：大數據簡介

Module 2：Hadoop Ecosystem介紹

Module 3：Hadoop 平台安裝

Module 4：Hadoop 分散式檔案系統（HDFS）

Module 5：Hadoop MapReduce

Module 6：Apache Hive

Module 7：Sqoop與Flume

Module 8：Apache Spark

Module 9：Spark 平台安裝

Module 10：RDD － Resilient distributed dataset

Module 11：Scala 程式開發基礎

Module 12：Spark SQL 及 DataFrame

Module 13：Spark 機器學習函式庫(MLlib)

Spark SQL 的由來

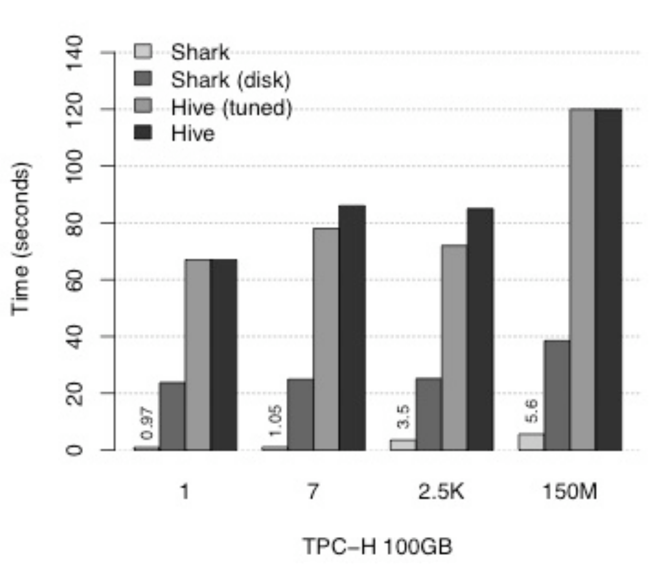
- Hive提供**熟悉SQL**但又不理解的MapReduce的技術人員快速上手的工具
- 但MapReduce計算過程中**大量的磁碟I / O造成效能瓶頸**，SQL-on-Hadoop的工具因應而生
  - MapR的Drill
  - Cloudera的Impala
  - **Shark(Spark SQL的前身)**
- Spark平台的SQL專案原為Shark，但Shark與Hive相依性太高；故另啟SparkSQL專案以取得更大彈性

Spark SQL 介紹

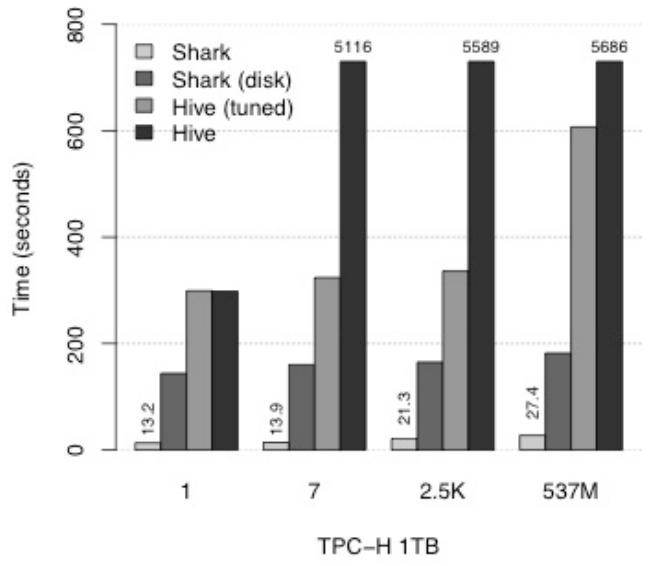
- Spark SQL is Apache Spark's module for working with **structured data**.
- **無縫與Spark程式整合**
- 提供**SQL-Like語法或DataFrame API**操作RDD
  - 降低學習門檻(相較Scala RDD API)
  - 提供更佳效能(In memory、SQL最佳化)

```
context = HiveContext(sc)
results = context.sql(
   "SELECT * FROM people")
names = results.map(p => p.name)
```

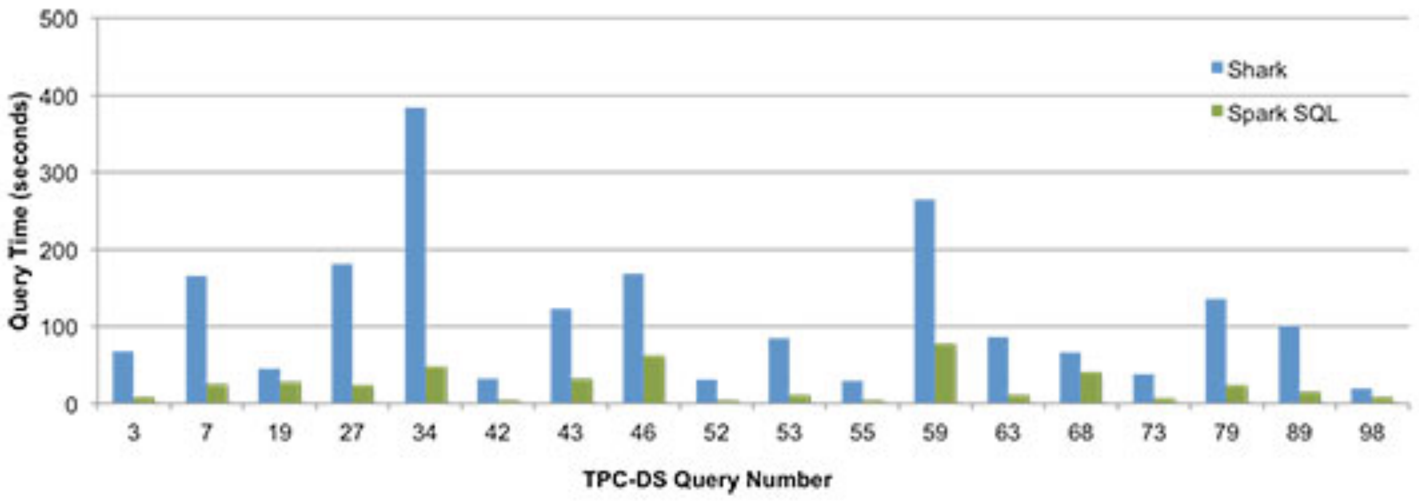
效能比較



Shark
Shark (disk)
Hive (tuned)
Hive
Time (seconds)
0.97
1.05
3.5
5.6
1
7
2.5K
150M
TPC–H 100GB



Shark
Shark (disk)
Hive (tuned)
Hive
Time (seconds)
5116
5589
5686
13.2
13.9
21.3
27.4
1
7
2.5K
537M
TPC–H 1TB

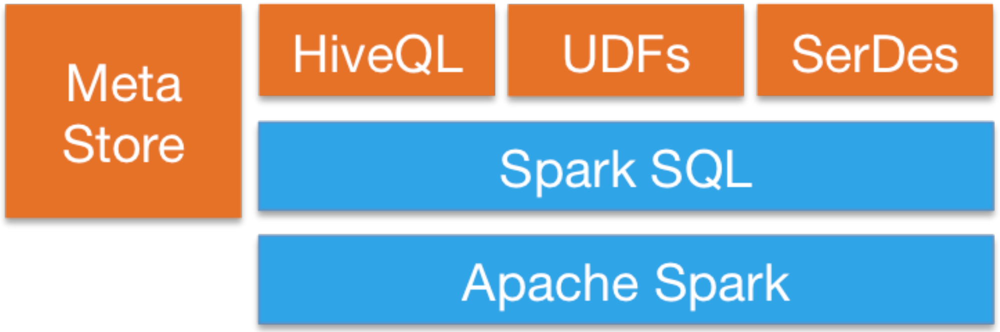


Shark
Spark SQL
Query Time (seconds)
3
7
19
27
34
42
43
46
52
53
55
59
63
68
73
79
89
98
TPC-DS Query Number

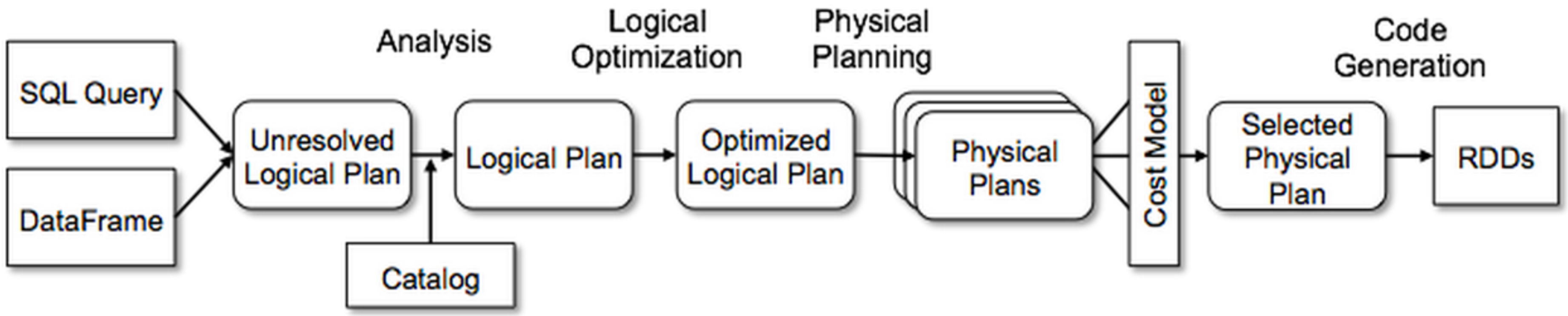
Spark SQL 介紹

- 提供**單一程式介面存取多種資料來源**
  - Hive、Parquet
  - JSON、JDBC
- 與既有Hive應用無縫接軌
  - 可存取Hive Table
  - UDFs、SerDes直接使用

```
context.jsonFile("s3n://…")
   .registerTempTable("json")
results = context.sql(
   """SELECT * FROM people JOIN json …""")
```



Spark SQL Query Planning



Spark SQL支援的資料來源

Built-In

{ JSON } JDBC Parquet

HIVE MySQL PostgreSQL

HDFS amazon web services S3 H2

External

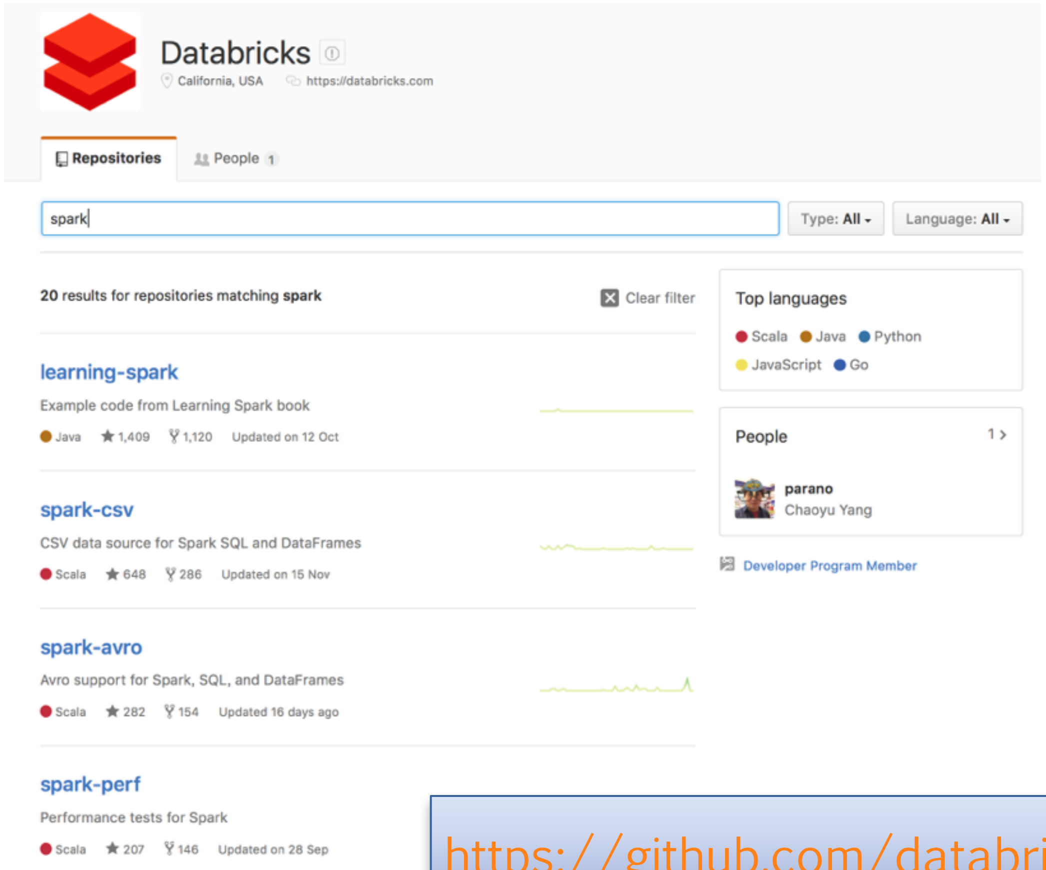
AVRO CSV dBase

APACHE HBASE elasticsearch. cassandra

amazon web services Amazon Redshift and more…

Spark SQL支援的資料來源

- External的資料來源可由Databricks的Github取得


Databricks
California, USA https://databricks.com

Repositories People 1

spark

Type: All Language: All

20 results for repositories matching spark

Clear filter

learning-spark
Example code from Learning Spark book
● Java ★ 1,409 1,120 Updated on 12 Oct

spark-csv
CSV data source for Spark SQL and DataFrames
● Scala ★ 648 286 Updated on 15 Nov

spark-avro
Avro support for Spark, SQL, and DataFrames
● Scala ★ 282 154 Updated 16 days ago

spark-perf
Performance tests for Spark
● Scala ★ 207 146 Updated on 28 Sep

Top languages
● Scala ● Java ● Python ● JavaScript ● Go

People 1
parano
Chaoyu Yang

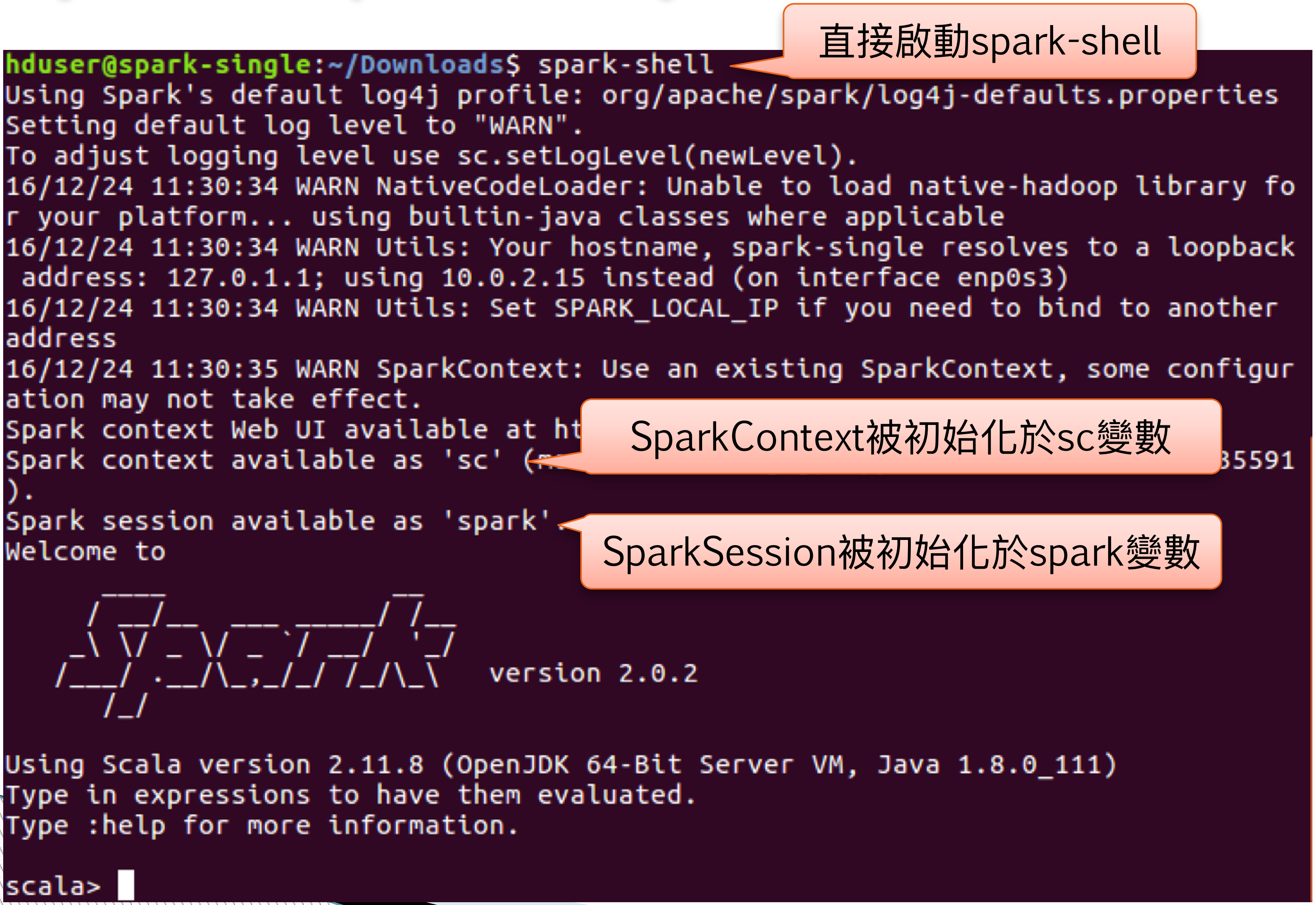
Developer Program Member

https://github.com/databricks

SparkSQL in Spark-Shell

- SparkSQL已內建於Spark平台中，不需Hive即可直接使用
- 在程式中，SparkSQL皆透過**SparkSession**操作
- 在spark-shell啟動後，兩個重要的類別會自動被初始化
  - **sc變數**：SparkContext的Instance，為操作RDD的接口
  - **spark變數**：SparkSession的Instance，為操作DataFram API及SQL的接口

SparkSQL in Spark-Shell



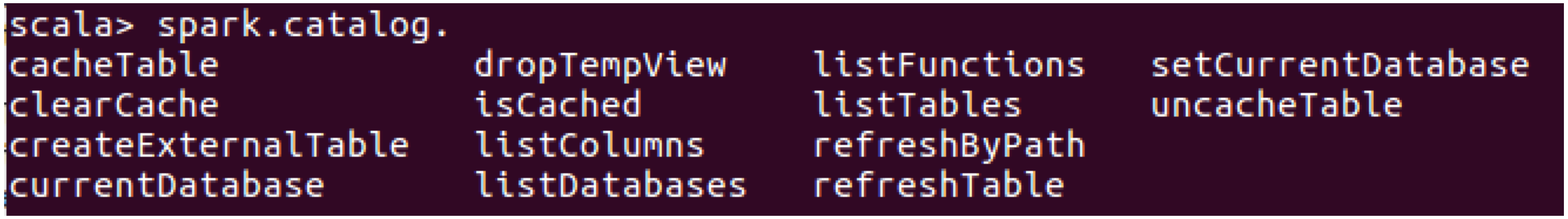
```
hduser@spark-single:~/Downloads$ spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
16/12/24 11:30:34 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
16/12/24 11:30:34 WARN Utils: Your hostname, spark-single resolves to a loopback
 address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
16/12/24 11:30:34 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
16/12/24 11:30:35 WARN SparkContext: Use an existing SparkContext, some configur
ation may not take effect.
Spark context Web UI available at ht
Spark context available as 'sc' (                                        35591
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.0.2
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_111)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 
```

SparkSession

- SparkSession作為SparkSQL的程式接口(以下簡稱為spark)，具有資料讀取、DataFrame操作及SQL指令操作功能
- spark.catalog
  - 新增及刪除SparkSQL中Table及DataBase的介面
    - ex：spark.catalog.listDatabases.show

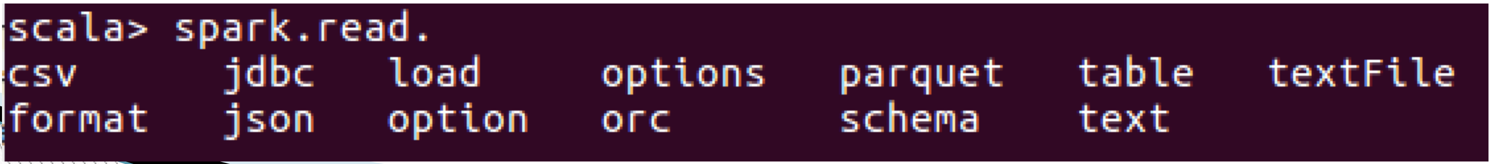


```
scala> spark.catalog.
cacheTable              dropTempView     listFunctions    setCurrentDatabase
clearCache              isCached         listTables       uncacheTable
createExternalTable     listColumns      refreshByPath
currentDatabase         listDatabases    refreshTable
```

Ref: http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.SparkSession

SparkSession

- spark.read
  - 讀取外部資料來源成為DataFrame
    - ex：spark.read.csv("file:/home/hduser/ratings.txt")
  - 內建支援的資料來源
    - csv、jdbc、parquet、text file、json
  - 以option設定讀取參數(如csv的欄位分割符號)
    - ex：spark.read.option("delimiter","\t").csv(path)



```
scala> spark.read.
csv       jdbc      load       options    parquet    table     textFile
format    json      option     orc        schema     text
```

SparkSession

- spark.sql
  - 以SQL指令操作SparkSQL的入口
    - ex：spark.sql("show databases").show
  - 在spark-shell中可省略spark.簡寫為sql("...")
  - SparkSQL的sql語法為SQL-92的子集
    - 支援project、where、order、join、group、having等語法

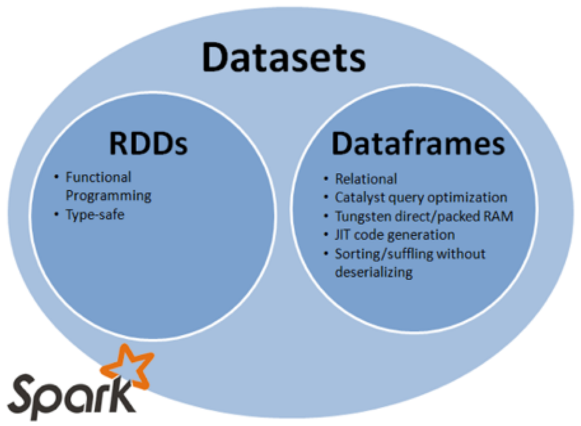
```
SELECT [DISTINCT] [column names]|[wildcard]
FROM [keyspace name.]table name
[JOIN clause table name ON join condition]
[WHERE condition]
[GROUP BY column name]
[HAVING conditions]
[ORDER BY column names [ASC | DSC]]
```

SparkSQL with Hive in Spark-Shell

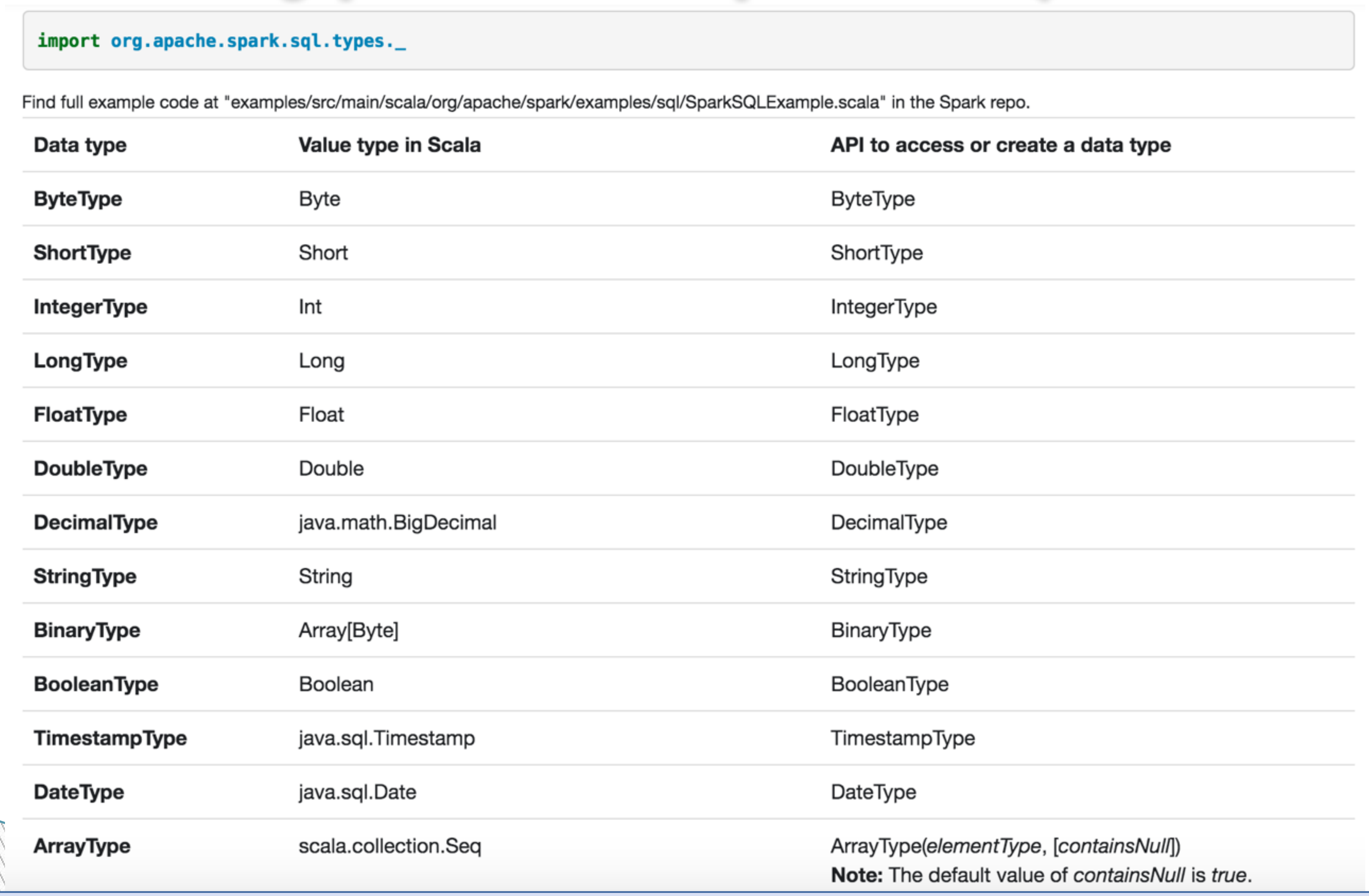
- SparkSQL支援讀取及寫入Hive的Database及Table
- Spark 2.0.2的SparkSQL最高支援Hive 1.2.1版本
  - 使用更高的Hive版本會產生不相容錯誤
- 要在Spark-shell中存取Hive需額外進行以下步驟
  - 將以下3個檔案copy至/usr/local/Spark/conf下
    - hive-site.xml (位於/usr/local/hive/conf)
    - core-site.xml (位於/usr/local/hadoop/etc/hadoop)
    - hdfs-site.xml (位於/usr/local/hadoop/etc/hadoop)
  - 啟動spark-shell時要引入mysql connector的jar檔
    - spark-shell --jars /usr/local/hive/lib/mysql-connector-java-5.1.40-bin.jar

DataSet & DataFrame

- DataSet在SparkSQL中是裝載data的類別
  - 由RDD衍生而來，承襲RDD的優點
    - strong typing、支援lambda操作
  - 可與RDD相互轉換
  - 支援RDD的transformation操作(map, flatMap, filter, etc.)
- DataFrame是具有欄位名稱的DataSet
- DataFrame將是Spark MLlib未來唯一支援的輸入型態


Datasets
RDDs
• Functional Programming
• Type-safe
Dataframes
• Relational
• Catalyst query optimization
• Tungsten direct/packed RAM
• JIT code generation
• Sorting/suffling without deserializing
Spark

DataTypes in SparkSQL



```
import org.apache.spark.sql.types._
```

Find full example code at "examples/src/main/scala/org/apache/spark/examples/sql/SparkSQLExample.scala" in the Spark repo.

Data type	Value type in Scala	API to access or create a data type
ByteType	Byte	ByteType
ShortType	Short	ShortType
IntegerType	Int	IntegerType
LongType	Long	LongType
FloatType	Float	FloatType
DoubleType	Double	DoubleType
DecimalType	java.math.BigDecimal	DecimalType
StringType	String	StringType
BinaryType	Array[Byte]	BinaryType
BooleanType	Boolean	BooleanType
TimestampType	java.sql.Timestamp	TimestampType
DateType	java.sql.Date	DateType
ArrayType	scala.collection.Seq	ArrayType(*elementType*, [*containsNull*]) **Note:** The default value of *containsNull* is *true*.

http://spark.apache.org/docs/latest/sql-programming-guide.html#data-types

Input & Output

- 透過sparkSession.read系列的方法來讀取不同資料來源
  - Local File System：讀取路徑加上**file:**的前置詞
    - ex：file:/usr/local/spark/examples/src/main/resources/people.json
  - HDFS：讀取路徑**不加file:**的前置詞
    - ex：/user/data/people.json
- 透過sparkSession.write系列的方法來輸出至不同目的地
  - write.save方法預設輸出為Parquet檔案
  - Local File System：讀取路徑加上**file:**的前置詞
  - HDFS：讀取路徑**不加file:**的前置詞

使用者自訂函式(UDF)

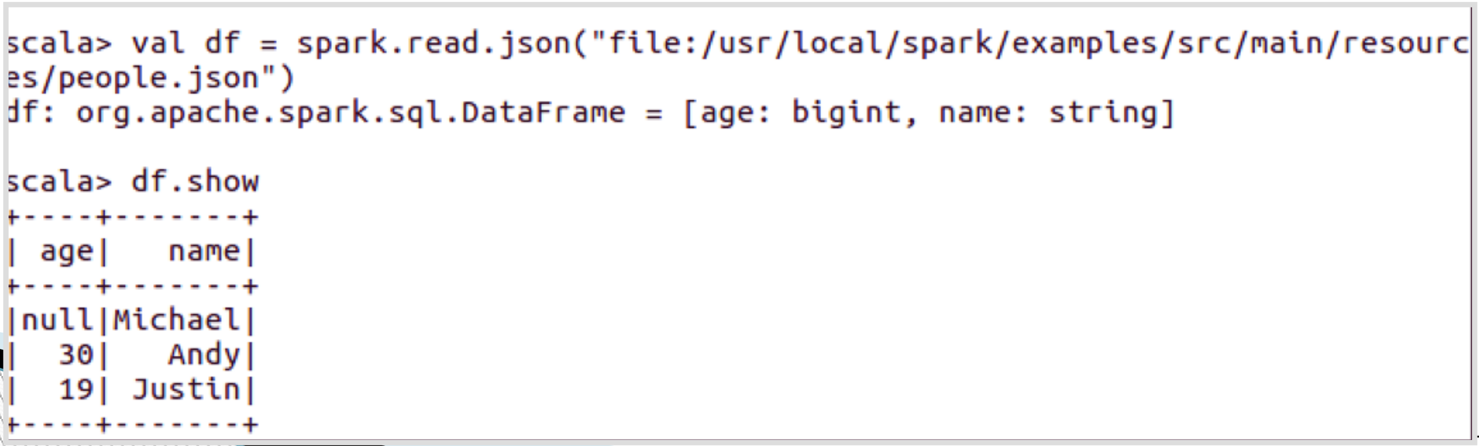
- 當DataFrame及SQL功能不敷使用時，可透過定義UDF來進行功能擴充
  - 以sparkSession.udf.register註冊UDF
  - 以lambda語法撰寫UDF
  - 直接在DataFrame API或SQL描述中使用

```
spark.udf.register("ageType", (age: Int)=> {
                var aType = "unknown"
                if (age > 0 && age < 20) aType = "teen"
                if (age >=20 && age < 40) aType = "adult"
                aType })

sql("select name, ageType(age) as ageType from people").show
```

從作中學：JSON & DataFram API

- 在spark-shell中輸入以下指令
  - val df = spark.read.json("file:/usr/local/spark/examples/src/main/resources/people.json")
    - 觀察shell中顯示df包含的欄位及型態
  - df.show()
    - 查看people.json的內容



```
scala> val df = spark.read.json("file:/usr/local/spark/examples/src/main/resourc
es/people.json")
df: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> df.show
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

從作中學：DataFram API

- df.select($"name").show
  - 只取name欄位
- df.select($"name", $"age" + 1).show()
  - 對欄位作變化(age=age+1)
- df.select($"name").filter($"age" > 10).show()
  - 篩選年齡大於10歲的人名
- df.groupBy($"age").count().show()
  - groupBy及count的使用
- df.groupBy($"age" > 10).count().show()
  - groupBy的進階應用

Ref: http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes

從作中學：SQL語法

- df.createOrReplaceTempView("people")
  - 將DataFrame註冊為view
- sql("select name from people").show
  - 只取name欄位
- sql("select name, age+1 from people").show
  - 對欄位作變化(age=age+1)
- sql("select name from people where age > 10").show
  - 篩選年齡大於10歲的人名
- sql("select age,count(*) from people group by age").show
  - groupBy及count的使用
- sql("select age > 10,count(*) from people group by age>10").show
  - groupBy及count的進階應用

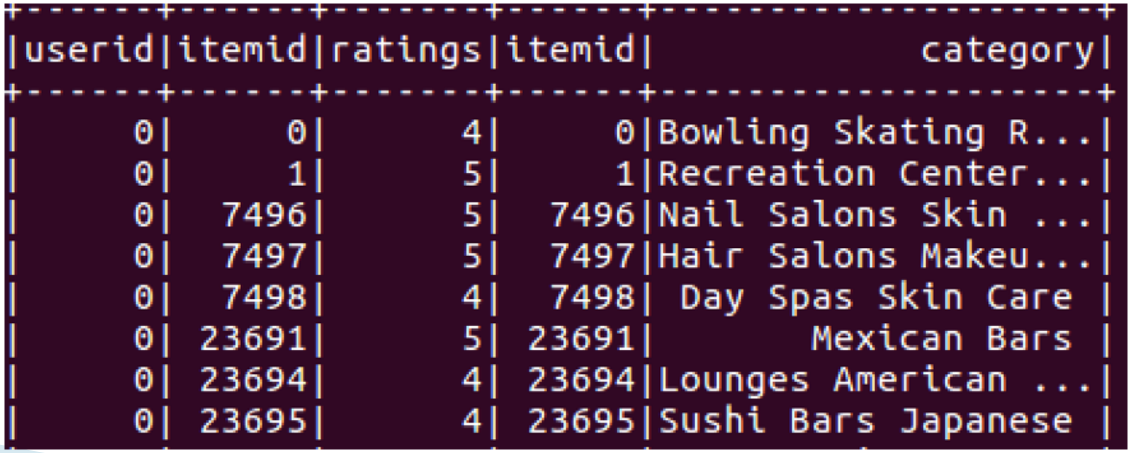
從作中學：UDF

- 定義ageType：傳入age，0~20歲為teenager、20~40為adult、其餘為unknown
- 定義toUpper：傳入name，將name轉為大寫
- sql("select toUpper(name) as name, ageType(age) as ageType) from people")

```
spark.udf.register("ageType", (age: Int)=> {
                var aType = "unknown"
                if (age > 0 && age < 20) aType = "teen"
                if (age >=20 && age < 40) aType = "adult"
                aType })
spark.udf.register("toUpper", (name: String)=> name.toUpperCase)
sql("select toUpper(name), ageType(age) as ageType from people").show
```

實作演練：不同來源資料的JOIN

- 在Hive課程中我們已在Hive環境中建立Yelp的items資料表，接下來我們透過SparkSQL讀取ratings.txt檔案，並在SparkSQL中建立ratings View
- 利用SQL語法篩選出userid=0的使用者且評價>=4的itemid及category



```
+------+------+-------+------+--------------------+
|userid|itemid|ratings|itemid|            category|
+------+------+-------+------+--------------------+
|     0|     0|      4|     0|Bowling Skating R...|
|     0|     1|      5|     1|Recreation Center...|
|     0|  7496|      5|  7496|Nail Salons Skin ...|
|     0|  7497|      5|  7497|Hair Salons Makeu...|
|     0|  7498|      4|  7498| Day Spas Skin Care |
|     0| 23691|      5| 23691|        Mexican Bars |
|     0| 23694|      4| 23694|Lounges American ...|
|     0| 23695|      4| 23695|Sushi Bars Japanese |
```

實作演練：不同來源資料的JOIN

- 啟動spark-shell
  - 注意要完成spark與hive的連結設定，並載入mysql-connector的jar
- 輸入以下指令
  - val path="file:/home/hduser/Downloads/ratings.txt"
  - val ratingDF1=spark.read.option("header","true").option("inferSchema","true").option("delimiter",":").csv(path) #讀入ratings.txt為DF
  - ratingDF1.select("userid","itemid","ratings").createOrReplaceTempView("ratings") #在SparkSQL中建立VIEW
  - sql("show tables").show #顯示目前SparkSQL中的資料表
  - val userDf=sql("select userid,i.itemid,ratings,category from ratings r join items i on r.itemid=i.itemid where userid=0 and r.ratings>=4")
    - 以itemid為key作join，篩選出userid=0的使用者且評價>=4的資料
  - userDf.show

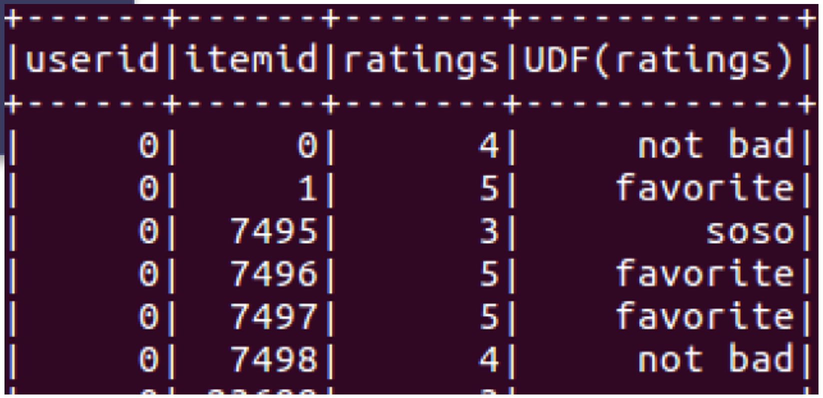
實作演練：DataFrame內容的輸出

- 接續上頁實作
  - userDf.write.save("file:/home/hduser/Downloads/userDf.parquet") #存成parquet檔案
  - userDf.write.csv("file:/home/hduser/Downloads/userDf.csv") #存成csv檔案
  - userDf.write.json("file:/home/hduser/Downloads/userDf.json) #存成json檔案
  - userDf.write.save("/user/userDf.parquet") #存成parquet檔案，並寫入HDSF

實作演練：UDF

- 接續上頁實作
- 定義favorType：傳入ratings，5分為favorite、4分為not bad、其餘為soso
- sql("select *, favorType(ratings) from ratings where userid=0").show

```
spark.udf.register("favorType", (ratings: Int)=> {
    var fType = "soso"
    if (ratings == 5) fType = "favorite"
    if (ratings == 4) fType = "not bad"
    fType })
```



```
+------+------+-------+------------+
|userid|itemid|ratings|UDF(ratings)|
+------+------+-------+------------+
|     0|     0|      4|     not bad|
|     0|     1|      5|    favorite|
|     0|  7495|      3|        soso|
|     0|  7496|      5|    favorite|
|     0|  7497|      5|    favorite|
|     0|  7498|      4|     not bad|
```