

# Spark SQL

## DataFrame

**A distributed collection of rows organised into named columns.**

**An abstraction for selecting, filtering, aggregating, and plotting structured data.**

**Previously => SchemaRDD**

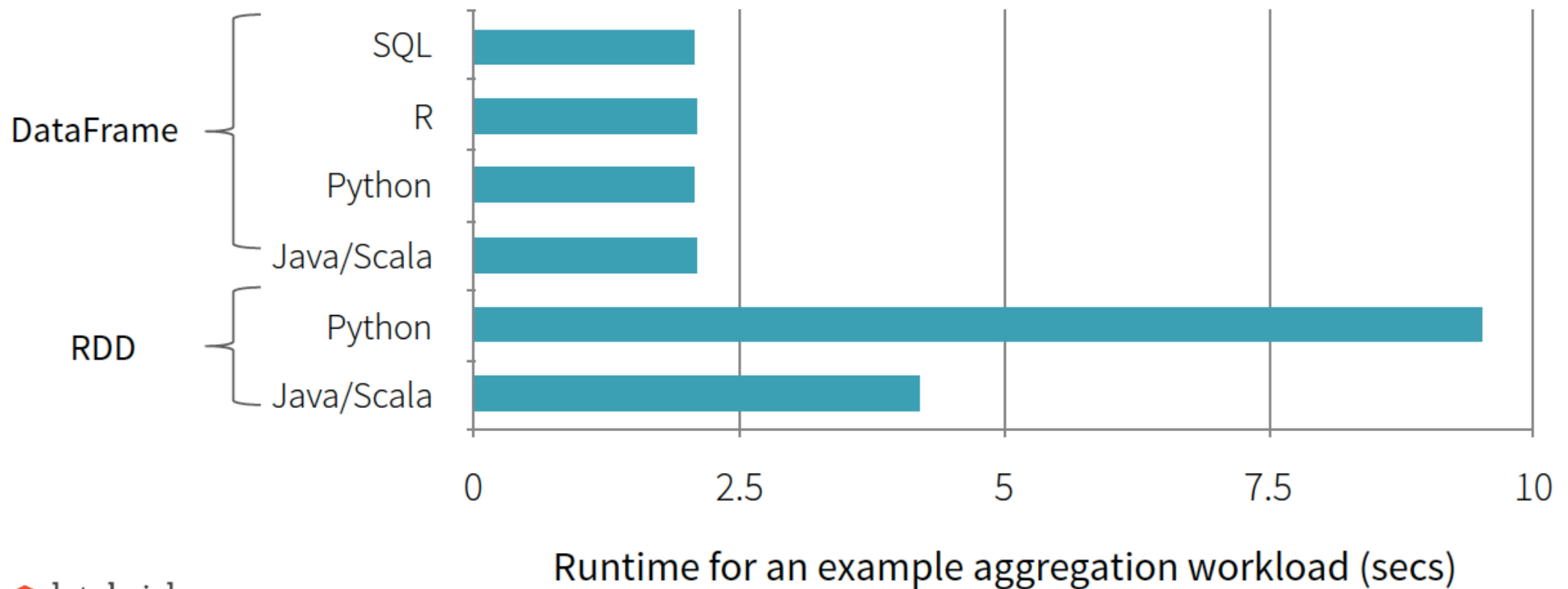
**Creating and running Spark program faster**

- Write less code**
- Read less data**
- Let the optimizer do the hard work**



**Spark SQL**  
is about more than SQL.

# Benefit of Logical Plan: Performance Parity Across Languages



# SparkSQL can leverage the Hive metastore

Hive Metastore can also be leveraged by a wide array of applications

- Spark
- Hive
- Impala

Available from HiveContext

```
context = ps.HiveContext(sc)

# query with SQL
results = context.sql(
    "SELECT * FROM people")

# apply Python transformation
names = results.map(lambda p: p.name)
```

Spark SQL

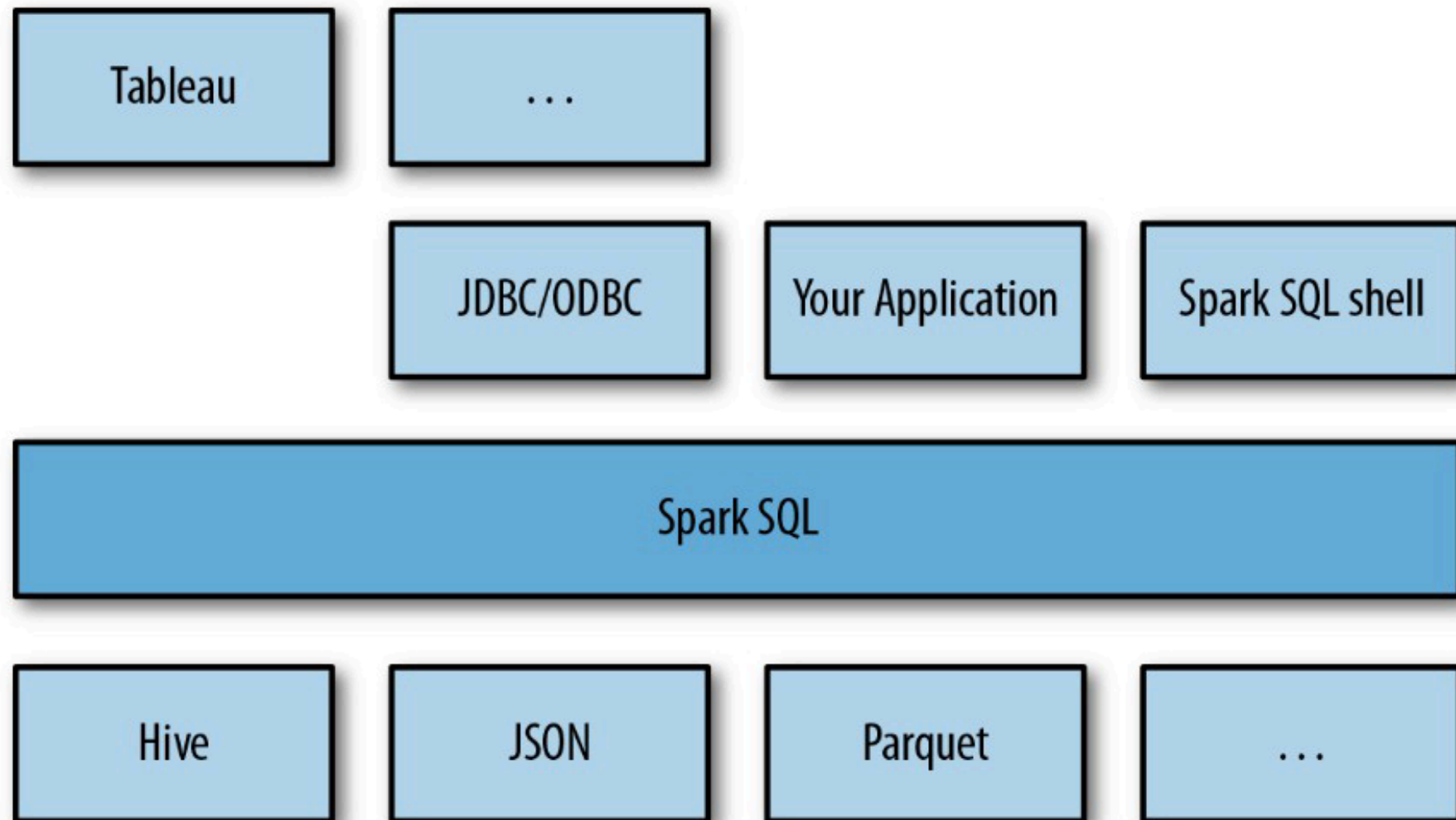
Spark Core

## Unified interface for structured data



Image credit: <http://barrymieny.deviantart.com/>

# Spark SQL usage



# Link Hive Metastore with Spark-Shell

```
$ spark-shell --jars mysql-connector-java-5.1.23.jar
```

```
scala > val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
```

```
scala> sqlContext.sql("CREATE TABLE IF NOT EXISTS movie(userid  
STRING, movieid STRING, rating INT, timestamp STRING) ROW FORMAT  
DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'")
```

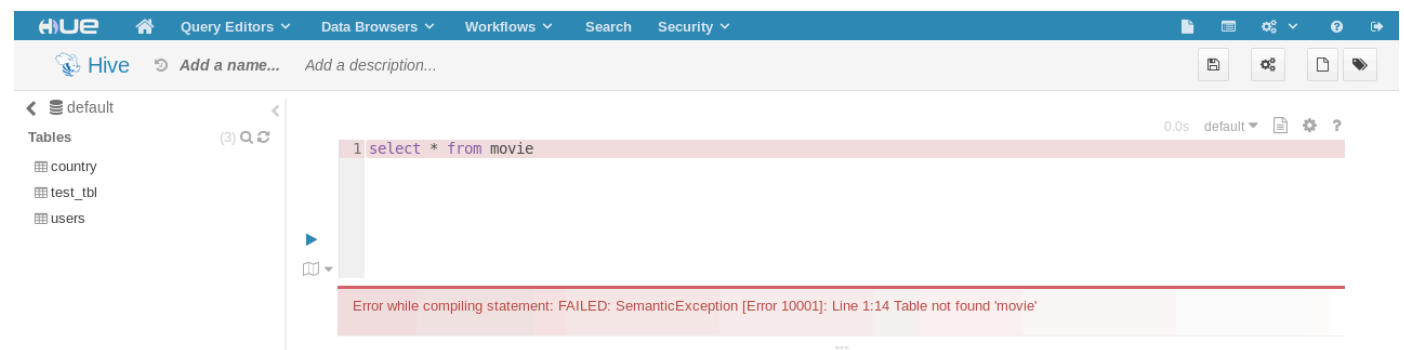
```
scala> sqlContext.sql("LOAD DATA LOCAL INPATH '/home/cloudera/  
movielens_dataset/ml-100k/u.data' INTO TABLE movie")
```

```
scala> val result = sqlContext.sql("SELECT * FROM movie")
```

```
scala> result.show()
```

```
scala> result.show()
```

userid	movieid	rating	timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488



# Link Hive Metastore with Spark-Shell

Copy the configuration file

```
$sudo cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/
```

```
$ spark-shell
```

```
scala > val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
```

```
scala> sqlContext.sql("CREATE TABLE IF NOT EXISTS movie(userid  
STRING, movieid STRING, rating INT, timestamp STRING) ROW FORMAT  
DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n')
```

```
scala> sqlContext.sql("LOAD DATA LOCAL INPATH '/home/cloudera/  
movielens_dataset/ml-100k/u.data' INTO TABLE movie")
```

```
scala> val result = sqlContext.sql("SELECT * FROM movie")
```

```
scala> result.show()
```



## HUE File Browser

Search for file name  ⚙️ Actions ✖️ Move to trash 📁 Upload ➕ New

Home / user / hive / warehouse ▼ History 🗑️ Trash

<input type="checkbox"/>	Name	Size	User	Group	Permissions	Date
<input type="checkbox"/>	↑		hive	supergroup	drwxrwxrwx	August 10, 2016 01:09 PM
<input type="checkbox"/>	.		hive	supergroup	drwxrwxrwx	October 14, 2016 02:56 AM
<input type="checkbox"/>	country		cloudera	supergroup	drwxrwxrwx	October 13, 2016 08:38 AM
<input type="checkbox"/>	movie		cloudera	supergroup	drwxrwxrwx	October 14, 2016 03:04 AM
<input type="checkbox"/>	test_tbl		cloudera	supergroup	drwxrwxrwx	October 13, 2016 02:43 AM

## HUE Metastore Manager

default

Tables

- country
- movie
- test\_tbl
- users

(4) 🔍 ↻

Databases > default



### STATS

Default Hive database

public (ROLE)

Location

### TABLES

Search for a table...

View

Browse Data

Drop

<input type="checkbox"/>	Table Name	Comment	Type
<input type="checkbox"/>	country	Imported by sqoop on 2016/10/13 08:38:37	Table
<input type="checkbox"/>	movie		Table
<input type="checkbox"/>	test_tbl		Table
<input type="checkbox"/>	users		Table

# Spark SQL Meals Data

## Upload a data to HDFS

```
$ wget https://github.com/bobbylovemovie/trainbigdata/raw/  
master/Spark/events.txt
```

```
$ wget https://github.com/bobbylovemovie/trainbigdata/raw/  
master/Spark/meals.txt
```

```
$ hadoop fs -put events.txt /user/cloudera/input
```

```
$ hadoop fs -put meals.txt /user/cloudera/input
```

# Spark SQL : Preparing data

**\$ pyspark**

```
>>> meals_rdd = sc.textFile("hdfs:///user/cloudera/input/meals.txt")
>>> events_rdd = sc.textFile("hdfs:///user/cloudera/input/events.txt")
>>> header_meals = meals_rdd.first()
>>> header_events = events_rdd.first()
>>> meals_no_header = meals_rdd.filter(lambda row: row != header_meals)
>>> events_no_header = events_rdd.filter(lambda row: row != header_events)
>>> meals_json = meals_no_header.map(lambda
row: row.split(';')).map(lambda row_list: dict(zip(header_meals.split(';'),
row_list)))
>>> events_json = events_no_header.map(lambda
row: row.split(';')).map(lambda row_list: dict(zip(header_events.split(';'),
row_list)))
```

```
>>> import json
>>> def type_conversion(d, columns) :
...     for c in columns:
...         d[c] = int(d[c])
...     return d
...
>>> meal_typed = meals_json.map(lambda
j:json.dumps(type_conversion(j, ['meal_id','price'])))
>>> event_typed = events_json.map(lambda
j:json.dumps(type_conversion(j, ['meal_id','userid'])))
```

## Spark SQL : Create DataFrame

```
>>> meals_dataframe = sqlContext.jsonRDD(meal_typed)
>>> events_dataframe = sqlContext.jsonRDD(event_typed)
>>> meals_dataframe.head()
|Row(dt=u'2013-01-01', meal_id=1, price=10, type=u'french')
>>> meals_dataframe.printSchema()
```

```
root
 |-- dt: string (nullable = true)
 |-- meal_id: long (nullable = true)
 |-- price: long (nullable = true)
 |-- type: string (nullable = true)
```

## Running SQL Query

```
>>> meals_dataframe.registerTempTable('meals')
```

```
>>> events_dataframe.registerTempTable('events')
```

```
>>> sqlContext.sql("SELECT * FROM meals LIMIT 5").collect()
```

```
[Row(dt=u'2013-01-01', meal_id=1, price=10, type=u'french'), Row(dt=u'2013-01-01', meal_id=2, price=13, type=u'chinese'), Row(dt=u'2013-01-02', meal_id=3, price=9, type=u'mexican'), Row(dt=u'2013-01-03', meal_id=4, price=9, type=u'italian'), Row(dt=u'2013-01-03', meal_id=5, price=12, type=u'chinese')]
```

```
>>> meals_dataframe.take(5)
```

```
[Row(dt=u'2013-01-01', meal_id=1, price=10, type=u'french'), Row(dt=u'2013-01-01', meal_id=2, price=13, type=u'chinese'), Row(dt=u'2013-01-02', meal_id=3, price=9, type=u'mexican'), Row(dt=u'2013-01-03', meal_id=4, price=9, type=u'italian'), Row(dt=u'2013-01-03', meal_id=5, price=12, type=u'chinese')]
```

## Spark SQL : More complex query

```
>>> sqlContext.sql("""  
    SELECT type, COUNT(type) AS cnt FROM  
    meals  
    INNER JOIN  
    events on meals.meal_id = events.meal_id  
    WHERE  
    event = 'bought'  
    GROUP BY  
    type  
    ORDER BY cnt DESC  
    """).collect()
```

```
[Row(type=u'italian', cnt=22575), Row(type=u'french', cnt=16179), Row(type=u'mex  
ican', cnt=8792), Row(type=u'japanese', cnt=6921), Row(type=u'chinese', cnt=6267  
) , Row(type=u'vietnamese', cnt=3535)]
```