Spark SQL

Create and run Spark programs faster:

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

DataFrame

noun – [dey-tuh-freym]

1. A distributed collection of rows organized into named columns.
2. An abstraction for selecting, filtering, aggregating and plotting structured data (*cf. R, Pandas*).
3. Archaic: Previously SchemaRDD (*cf. Spark* < *1.3*).

Write Less Code: Input & Output

Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \

  .saveAsTable("fasterData")
```

databricks

Write Less Code: Input & Output

Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \

  .saveAsTable("fasterData")
```

read and write functions create new builders for doing I/O

Write Less Code: Input & Output

Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \

  .saveAsTable("fasterData")
```

Builder methods specify:

- Format
- Partitioning
- Handling of existing data

Write Less Code: Input & Output

Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \

  .saveAsTable("fasterData")
```

`load(…)`, `save(…)` or `saveAsTable(…)` finish the I/O specification

Write Less Code: Data Source API

Spark SQL's Data Source API can read and write DataFrames using a variety of formats.

Built-In

{ JSON } JDBC Parquet

HIVE MySQL PostgreSQL

HDFS amazon web services S3 H2

plain text* HIVE ORC

External

AVRO CSV dBase

APACHE HBASE elasticsearch. cassandra

amazon web services Amazon Redshift memsql and more…

Find more sources at http://spark-packages.org/

Write Less Code: High-Level Operations

Solve common problems concisely using DataFrame functions:

- Selecting columns and filtering
- Joining different data sources
- Aggregation (count, sum, average, etc)
- Plotting results with Pandas

Write Less Code: Compute an Average

Using RDDs

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
   .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
   .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
   .collect()
```

Using SQL

```
SELECT name, avg(age)
FROM people
GROUP BY name
```

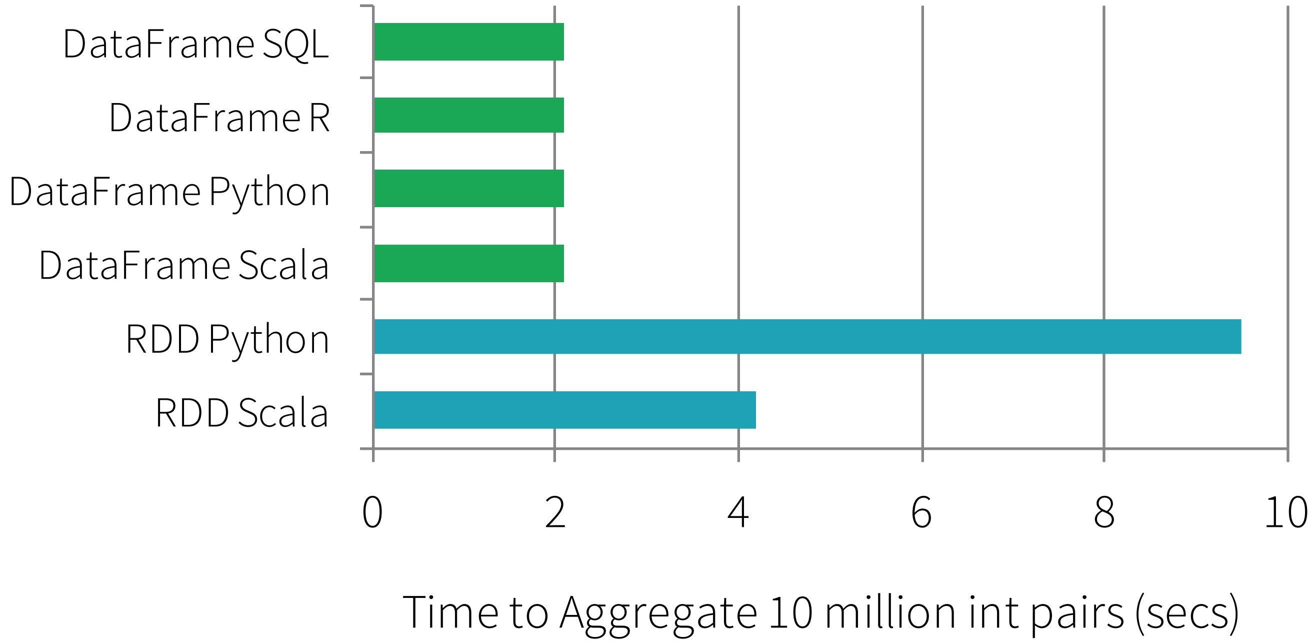
Using DataFrames

```
sqlCtx.table("people") \
   .groupBy("name") \
   .agg("name", avg("age")) \
   .map(lambda …) \
   .collect()
```

Full API Docs

- Python
- Scala
- Java
- R

Not Just Less Code, Faster Too!



Seamlessly Integrated

Intermix DataFrame operations with
custom Python, Java, R, or Scala code

```
zipToCity = udf(lambda zipCode: <custom logic here>)

def add_demographics(events):
   u = sqlCtx.table("users")
   events \
     .join(u, events.user_id == u.user_id) \
     .withColumn("city", zipToCity(df.zip))
```

Augments any DataFrame that contains `user_id`

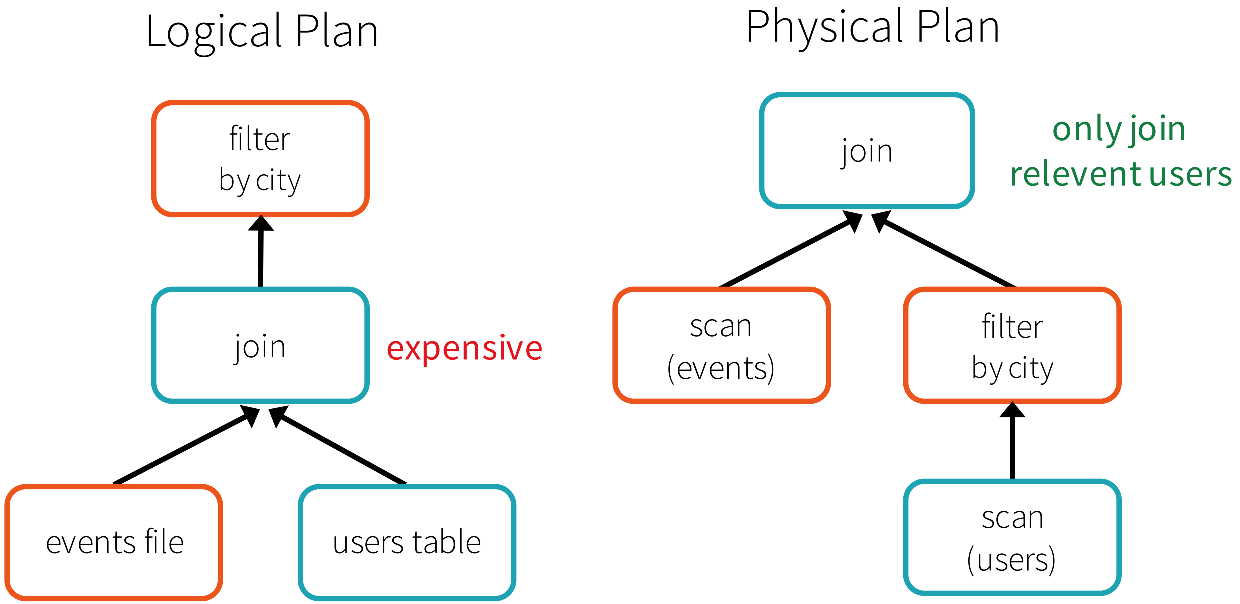
Optimize Full Pipelines

Optimization happens as late as possible, therefore Spark SQL can optimize *even across functions.*

```
events = add_demographics(sqlCtx.load("/data/events", "json"))

training_data = events \
  .where(events.city == "Amsterdam") \
  .select(events.timestamp) \
  .collect()
```

```python
def add_demographics(events):
   u = sqlCtx.table("users")                     # Load Hive table
   events \
     .join(u, events.user_id == u.user_id) \     # Join on user_id
     .withColumn("city", zipToCity(df.zip))      # Run udf to add city column
events = add_demographics(sqlCtx.load("/data/events", "json"))
training_data = events.where(events.city == "Amsterdam").select(events.timestamp).collect()
```



```
def add_demographics(events):
    u = sqlCtx.table("users")                       # Load partitioned Hive table
    events \
      .join(u, events.user_id == u.user_id) \       # Join on user_id
      .withColumn("city", zipToCity(df.zip))        # Run udf to add city column
events = add_demographics(sqlCtx.load("/data/events", "parquet"))
training_data = events.where(events.city == "Amsterdam").select(events.timestamp).collect()
```