Spark SQL: Relational Data Processing in Spark (SIGMOD 2015)

Presented by Ankur Dave
CS294-110, Fall 2015
Problem

Imperative ⇒ Declarative

Semi-Structured Data & Advanced Analytics

⇒

No support in existing systems
Spark SQL

1. DataFrame API

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

2. Catalyst Optimizer
DataFrames

- Language-integrated declarative API

```scala
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()
```

- Schema inference for semi-structured data

- Allows direct access to JVM objects
  - Unlike other declarative systems

- Supports complex types like vectors
  - Easy to add new types

```scala
sqlCtx.table("people") \ .groupBy("name") \ .agg("name", avg("age")) \ .collect()
```
Catalyst Optimizer

• Rules written using Scala pattern matching
  • Allows writing complex rules
  • Example: join elimination

• Exploits structure of data source
  • Example: predicate pushdown for Parquet
  • Easy to add new data sources (e.g., Succinct)

• Code generation for expression evaluation

• Cost-based join algorithm selection (shuffle vs. BitTorrent broadcast)
Evaluation

- Gains over Spark and Shark due to code generation
  - Optimizer less mature than Impala’s

- More important: ease of use and extensibility
<table>
<thead>
<tr>
<th>Why Spark SQL?</th>
<th>Easy to Program?</th>
<th>Advanced Analytics?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Spark</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SHARK, Hive</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>cloudera, IMPALA</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Spark SQL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Flink, DryadLINQ</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Lasting Impact?

• Extensible optimizer

```scala
// Push down filter through EXCEPT
case Filter(condition, e @ Except())
  val (deterministic, nondeterministic)
val rewrites = buildRewrites(e)
Filter(nondeterministic,
    Except(
sqlCtx.table("people")
  .groupBy("name")
  .agg("name", avg("age"))
  .collect()
```

• Language integration