Spark SQL
The 8 fastest-growing tech skills worth over $110,000

No. 1: Spark, up 120%, worth $113,214
DO you know how to write code in Spark?
Can you write SQL?

“SQL is a highly sought-after technical skill due to its ability to work with nearly all databases.”

Ibro Palic, CEO of Resumes Templates
History and Evolution of Big Data Technologies

Procedural Programming interface

Declarative Queries

Automatic Optimization
So Far...

- We have established that we need a platform with Automatic Optimization.
What user want?

1. ETL from different sources
2. Advanced Analytics
Introducing Spark SQL: Relational Data Processing in Spark
Background

- Apache Spark is a **general-purpose cluster computing engine** with APIs in Scala, Java and Python and libraries for streaming, graph processing and machine learning.
- RDDs are **fault-tolerant**, in that the system can recover lost data using the lineage graph of the RDDs (by rerunning operations such as the filter above to rebuild missing partitions). They can also explicitly be cached in memory or on disk to support iteration.
- Shark, a modified the Apache Hive system to run on Spark and implemented traditional RDBMS optimizations, such as columnar processing, over the Spark engine.
Goals for Spark SQL

- Support **Relational Processing** both within Spark programs and on external data sources.
- Provide **High Performance** using established DBMS techniques.
- Easily support **New Data Sources**.
- Enable **Extension** with advanced analytics algorithms such as graph processing and machine learning.
Programming Interface

- JDBC
- Console
- User Programs (Java, Scala, Python)
- Spark SQL
- DataFrame API
- Catalyst Optimizer
- Spark
- Resilient Distributed Datasets
DataFrame API

- DataFrame is a distributed collection of rows with a homogeneous schema

```scala
ctx = new HiveContext()
users = ctx.table("users")
young = users.where(users("age") < 21)
println(young.count())
```
Data Model and DataFrame Operations

- Spark SQL uses a nested data model based on Hive
- It supports all major SQL data types, including boolean, integer, double, decimal, string, date, timestamp and also User Defined Data types

Example of DataFrame Operations

```python
employees
  .join(dept, employees("deptId") == dept("id"))
  .where(employees("gender") == "female")
  .groupBy(dept("id"), dept("name"))
  .agg(count("name"))
```
DataFrame Operations Cont.

```python
users.where(users("age") < 21)
    .registerTempTable("young")
ctx.sql("SELECT count(*), avg(age) FROM young")
```

#Access DF with DSL or SQL
Real World Problems

#Heterogeneous Data Sources
Schema Inference

- Spark SQL can automatically infer the schema of these objects using **reflection**
- **Scala/Java** - extracted from the language’s type system
- **Python** – Sampling the Dataset
In – Memory Caching

#Invoked with .cache()
User-Defined Functions

How Spark SQLs User defined functions are different than traditional Database Systems?
Catalyst Optimizer

- Catalyst is based on functional programming constructs in Scala

  - Ability to add new optimization techniques and features to Spark SQL
  - Ability to extend the optimizer
Catalyst Optimization

#Trees

#Rules

```java
    tree.transform {
        case Add(Literal(c1), Literal(c2)) => Literal(c1+c2)
    }
```
Catalyst Optimization Cont.

- Rule Based Optimization
- Cost Based Optimization
Query Planning in Spark SQL

Figure 3: Phases of query planning in Spark SQL. Rounded rectangles represent Catalyst trees.
Extension Points

#Open Source Projects
Extension Points Cont.

- Data Sources
  - Examples:
    - CSV
    - Avro
    - Parquet
    - JDBC
Extension Points Cont.

- **User Defined Types (UDTs)**

```scala
class PointUDT extends UserDefinedType[Point] {
  def dataType = StructType(Seq(  // Our native structure
    StructField("x", DoubleType),
    StructField("y", DoubleType)
  ))
  def serialize(p: Point) = Row(p.x, p.y)
  def deserialize(r: Row) =
    Point(r.getDouble(0), r.getDouble(1))
}
```

#Useful for Machine Learning
Advanced Analytics Features

1. Schema Inference for Semi-structured Data
2. Query Federation to External Databases
Advanced Analytics Features Cont.

3. Integration with Spark’s Machine Learning Library

```python
(data = <DataFrame of (text, label) records>)

tokenizer = Tokenizer()
    .setInputCol("text").setOutputCol("words")

tf = HashingTF()
    .setInputCol("words").setOutputCol("features")

lr = LogisticRegression()
    .setInputCol("features")

pipeline = Pipeline().setStages([tokenizer, tf, lr])

model = pipeline.fit(data)
```
Evaluation

- SQL Performance

Figure 8: Performance of Shark, Impala and Spark SQL on the big data benchmark queries [31].
Evaluation Cont.

DataFrames vs. Native Spark Code

```scala
sum_and_count = 
    data.map(lambda x: (x.a, (x.b, 1))) 
    .reduceByKey(lambda x, y: (x[0]+y[0], x[1]+y[1])) 
    .collect()
[(x[0], x[1][0] / x[1][1]) for x in sum_and_count]

In contrast, the same program can be written as a simple manipulation using the DataFrame API:

df.groupBy("a").avg("b")
```
Pipeline Performance

[Chart showing the performance comparison between SQL + Spark and DataFrame in terms of runtime (seconds). The chart indicates that SQL + Spark has a lower runtime compared to DataFrame, with specific runtime values for filter and word count operations.]
Applications

- Generalized Online Aggregation
- Computational Genomics
- List is infinite only limited by your imagination...
Conclusion

Our Final Hash Tags

#A Platform with
#Automatic optimization
#Complex pipelines that mix relational and complex analytics
#Large-scale data analysis
#Semi-structured data
#Data types for machine learning
#Extensible optimizer called Catalyst
#Easy to add Optimization rules, data sources and data types
Thank You