cloudera

Spark SQL and DataFrames

Chapter 17

201509

Course Chapters

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrames and SparkSQL

In this chapter you will learn

- **What Spark SQL is**
- **What features the DataFrame API provides**
- **How to create a SQLContext**
- **How to load existing data into a DataFrame**
- **How to query data in a DataFrame**
- **How to convert from DataFrames to Pair RDDs**

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Spark SQL and DataFrames	Distributed Data Processing with Spark

- **Spark SQL and the SQL Context**
- Creating DataFrames
- Transforming and Querying DataFrames
- Saving DataFrames
- DataFrames and RDDs
- Comparing Spark SQL, Impala and Hive-on-Spark
- Conclusion
- Homework: Use Spark SQL for ETL

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What is Spark SQL?

- **What is Spark SQL?**
  - Spark module for structured data processing
  - Replaces Shark (a prior Spark module, now deprecated)
  - Built on top of core Spark

- **What does Spark SQL provide?**
  - The DataFrame API – a library for working with data as tables
    - Defines DataFrames containing Rows and Columns
    - DataFrames are the focus of this chapter!
  - Catalyst Optimizer – an extensible optimization framework
  - A SQL Engine and command line interface

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

SQL Context

- **The main Spark SQL entry point is a SQL Context object**
  - Requires a SparkContext
  - The SQL Context in Spark SQL is similar to Spark Context in core Spark
- **There are two implementations**
  - **`SQLContext`**
    - basic implementation
  - **`HiveContext`**
    - Reads and writes Hive/HCatalog tables directly
    - Supports full HiveQL language
    - Requires the Spark application be linked with Hive libraries
    - Recommended starting with Spark 1.5

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating a SQL Context

- **SQLContext is created based on the SparkContext**

Python

```
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
```

Scala

```
import org.apache.spark.sql.SQLContext
val sqlCtx = new SQLContext(sc)
import sqlCtx._
```

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Spark SQL and DataFrames	Distributed Data Processing with Spark

- Spark SQL and the SQL Context
- **Creating DataFrames**
- Transforming and Querying DataFrames
- Saving DataFrames
- DataFrames and RDDs
- Comparing Spark SQL, Impala and Hive-on-Spark
- Conclusion
- Homework: Use Spark SQL for ETL

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrames

- **DataFrames are the main abstraction in Spark SQL**
  - Analogous to RDDs in core Spark
  - A distributed collection of data organized into named columns
  - Built on a base RDD containing `Row` objects

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating DataFrames

- **DataFrames can be created**
  - From an existing structured data source (Parquet file, JSON file, etc.)
  - From an existing RDD
  - By performing an operation or query on another DataFrame
  - By programmatically defining a schema

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating a DataFrame from a JSON File

Python

```python
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
peopleDF = sqlCtx.jsonFile("people.json")
```

Scala

```scala
val sqlCtx = new SQLContext(sc)
import sqlCtx._
val peopleDF = sqlCtx.jsonFile("people.json")
```

File: people.json

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30, "pcode":"94304"}
{"name":"Carla", "age":19, "pcode":"10036"}
{"name":"Diana", "age":46}
{"name":"Étienne", "pcode":"94104"}
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating a DataFrame from a Data Source

- **Methods on the SQLContext object**
- **Convenience functions**
  - `jsonFile(filename)`
  - `parquetFile(filename)`
- **Generic base function: `load`**
  - `load(filename,source)` – load **`filename`** of type **`source`** (default Parquet)
  - `load(source,options…)` – load from a source of type **`source`** using options
  - Convenience functions are implemented by calling **`load`**
    - `jsonFile("people.json")` = `load("people.json", "json")`

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Data Sources

- **Spark SQL 1.3 includes three data source types**
  - **`json`**
  - **`parquet`**
  - **`jdbc`**
- **You can also use third party data source libraries, such as**
  - Avro
  - HBase
  - CSV
  - MySQL
  - and more being added all the time

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Generic Load Function Example: JDBC

- **Example: Loading from a MySQL database**

```
val accountsDF = sqlCtx.load("jdbc",
   Map("url"-> "jdbc:mysql://dbhost/dbname?user=…&password=…",
       "dbtable" -> "accounts"))
```

```
accountsDF = sqlCtx.load(source="jdbc", \
  url="jdbc:mysql://dbhost/dbname?user=…&password=…", \
  dbtable="accounts”)
```

Warning: Avoid direct access to databases in production environments, which may overload the DB or be interpreted as service attacks

- Use Sqoop to import instead

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Generic Load Function Example: Third-party or Custom Sources

- **You can also use custom or third party data sources**
- **Example: Read from an Avro file using the `avro` source in the Databricks Spark Avro package**

```
$ spark-shell --packages com.databricks:spark-avro_2.10:1.0.0
> …
> val myDF =
sqlCtx.load("myfile.avro","com.databricks.spark.avro")
```

```
$ pyspark --packages com.databricks:spark-avro_2.10:1.0.0
> …
> myDF = sqlCtx.load("myfile.avro","com.databricks.spark.avro")
```

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Spark SQL and DataFrames | **Distributed Data Processing with Spark**

- Spark SQL and the SQL Context
- Creating DataFrames
- **Transforming and Querying DataFrames**
- Saving DataFrames
- DataFrames and RDDs
- Comparing Spark SQL, Impala and Hive-on-Spark
- Conclusion
- Homework: Use Spark SQL for ETL

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Basic Operations (1)

- **Basic Operations deal with DataFrame metadata (rather than its data), e.g.**
  - **`schema`** – returns a Schema object describing the data
  - **`printSchema`** – displays the schema as a visual tree
  - **`cache`** / **`persist`** – persists the DataFrame to disk or memory
  - **`columns`** – returns an array containing the names of the columns
  - **`dtypes`** – returns an array of (column-name,type) pairs
  - **`explain`** – prints debug information about the DataFrame to the console

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Basic Operations (2)

- **Example: Displaying column data types using `dtypes`**

```
> peopleDF = sqlCtx.jsonFile("people.json")
> for item in peopleDF.dtypes(): print item
('age', 'bigint')
('name', 'string')
('pcode', 'string’)
```

```
> val peopleDF = sqlCtx.jsonFile("people.json")
> people.dtypes.foreach(println)
(age,LongType)
(name,StringType)
(pcode,StringType)
```


© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Working with Data in a DataFrame

- **Queries – create a new DataFrame**
  - DataFrames are immutable
  - Queries are analogous to RDD transformations
- **Actions – return data to the Driver**
  - Actions trigger "lazy" execution of queries

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Actions

- **Some DataFrame actions**
  - **`collect`** – return all rows as an array of **`Row`** objects
  - **`take(n)`** – return the first **`n`** rows as an array of **`Row`** objects
  - **`count`** – return the number of rows
  - **`show(n)`** – display the first **`n`** rows (default=20)

```
> peopleDF.count()
5L

> peopleDF.show(3)
age  name    pcode
null Alice   94304
30   Brayden 94304
19   Carla   10036
```

```
> peopleDF.count()
res7: Long = 5

> peopleDF.show(3)
age  name    pcode
null Alice   94304
30   Brayden 94304
19   Carla   10036
```

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Queries (1)

- **DataFrame query methods return new DataFrames**
  - Queries can be chained like transformations
- **Some query methods**
  - `distinct` – returns a new DataFrame with distinct elements of this DF
  - `join` – joins this DataFrame with a second DataFrame
    - several variants for inside, outside, left, right, etc.
  - `limit` – a new DF with the first `n` rows of this DataFrame
  - `select` – a new DataFrame with data from one or more columns of the base DataFrame
  - `filter` – a new DataFrame with rows meeting a specified condition

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Queries (2)

- **Example: A basic query with limit**

```
> peopleDF.limit(3).show
```

```
> peopleDF.limit(3).show()
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036

Output of `show`

```
age  name    pcode
null Alice   94304
30   Brayden 94304
19   Carla   10036
```

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Query Strings (1)

- **Some query operations take strings containing simple query expressions**
  - Such as **`select`** and **`where`**
- **Example: `select`**

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

```
peopleDF.
  select("age")
```

age
null
30
19
46
null

```
peopleDF.
  select("name","age")
```

name	age
Alice	null
Brayden	30
Carla	19
Diana	46
Étienne	null

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Query Strings (2)

- **Example: `where`**

```
peopleDF.
  where("age > 21")
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

age	name	pcode
30	Brayden	94304
46	Diana	null

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Querying DataFrames using Columns (1)

- **Some DF queries take one or more *columns* or *column expressions***
  - Required for more sophisticated operations
- **Some examples**
  - **`select`**
  - **`sort`**
  - **`join`**
  - **`where`**

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Querying DataFrames using Columns (2)

- **In Python, reference columns by name using *dot notation***

```
ageDF = peopleDF.select(peopleDF.age)
```

- **In Scala, columns can be referenced in two ways**

```
val ageDF = peopleDF.select($"age")
```

  – *OR*

```
val ageDF = peopleDF.select(peopleDF("age"))
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

age
null
30
19
46
null

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Querying DataFrames using Columns (3)

- **Column references can also be *column expressions***

```
peopleDF.select(peopleDF.name,peopleDF.age+10)
```

```
peopleDF.select(peopleDF("name"),peopleDF("age")+10)
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

name	age+10
Alice	null
Brayden	40
Carla	29
Diana	56
Étienne	null

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Querying DataFrames using Columns (4)

- **Example: Sorting in by columns (descending)**

```
peopleDF.sort(peopleDF.age.desc())
```

```
peopleDF.sort(peopleDF("age").desc)
```

`.asc` and `.desc` are column expression methods used with `sort`

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

age	name	pcode
46	Diana	null
30	Brayden	94304
19	Carla	10036
null	Alice	94304
null	Étienne	94104

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

SQL Queries

- **Spark SQL also supports the ability to perform SQL queries**
  - First, register the DataFrame as a "table" with the SQL Context

```
peopleDF.registerTempTable("people")
sqlCtx.sql("""SELECT * FROM people WHERE name LIKE "A%" """)
```

```
peopleDF.registerTempTable("people")
sqlCtx.sql("""SELECT * FROM people WHERE name LIKE "A%" """)
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

age	name	pcode
null	Alice	94304

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Spark SQL and DataFrames | **Distributed Data Processing with Spark**

- Spark SQL and the SQL Context
- Creating DataFrames
- Transforming and Querying DataFrames
- **Saving DataFrames**
- DataFrames and RDDs
- Comparing Spark SQL, Impala and Hive-on-Spark
- Conclusion
- Homework: Use Spark SQL for ETL

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Saving DataFrames

- **Data in DataFrames can be saved to a data source**
  - Built in support for JDBC and Parquet File
    - **`createJDBCTable`** – create a new table in a database
    - **`insertInto`** – save to an existing table in a database
    - **`saveAsParquetFile`** – save as a Parquet file (including schema)
    - **`saveAsTable`** – save as a Hive table (HiveContext only)
  - Can also use third party and custom data sources
    - **`save`** – generic base function

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Spark SQL and DataFrames	Distributed Data Processing with Spark

- Spark SQL and the SQL Context
- Creating DataFrames
- Transforming and Querying DataFrames
- Saving DataFrames
- **DataFrames and RDDs**
- Comparing Spark SQL, Impala and Hive-on-Spark
- Conclusion
- Homework: Use Spark SQL for ETL

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrames and RDDs (1)

- **DataFrames are built on RDDs**
  - Base RDDs contain **`Row`** objects
  - Use **`rdd`** to get the underlying RDD

```
peopleRDD = peopleDF.rdd
```

peopleDF

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

peopleRDD

Row[null,Alice,94304]
Row[30,Brayden,94304]
Row[19,Carla,10036]
Row[46,Diana,null]
Row[null,Étienne,94104]

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrames and RDDs (2)

- **Row RDDs have all the standard Spark actions and transformations**
  - Actions – **`collect`**, **`take`**, **`count`**, etc.
  - Transformations – **`map`**, **`flatMap`**, **`filter`**, etc.
- **Row RDDs can be transformed into PairRDDs to use map-reduce methods**

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Working with Row Objects

- **The syntax for extracting data from Rows depends on language**
- **Python**
  - Column names are object attributes
    - **`row.age`** – return age column value from row
- **Scala**
  - Use Array-like syntax
    - **`row(0)`** – returns element in the first column
    - **`row(1)`** – return element in the second column
    - etc.
  - Use type-specific **`get`** methods to return typed values
    - **`row.getString(`*`n`*`)`** – returns n^{th} column as a String
    - **`row.getInt(n)`** – returns n^{th} column as an Integer
    - etc.

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Extracting Data from Rows

- **Extract data from Rows**

```
peopleRDD = peopleDF.rdd
peopleByPCode = peopleRDD \
   .map(lambda row(row.pcode,row.name)) \
   .groupByKey()
```

```
val peopleRDD = peopleDF.rdd
peopleByPCode = peopleRDD.
  map(row => (row(2),row(1))).
  groupByKey())
```

```
Row[null,Alice,94304]
Row[30,Brayden,94304]
Row[19,Carla,10036]
Row[46,Diana,null]
Row[null,Étienne,94104]
```

```
(94304,Alice)
(94304,Brayden)
(10036,Carla)
(null,Diana)
(94104,Étienne)
```

```
(null,[Diana])
(94304,[Alice,Brayden])
(10036,[Carla])
(94104,[Étienne])
```

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Converting RDDs to DataFrames

- **You can also create a DF from an RDD**
  - **`sqlCtx.createDataFrame(`*`rdd`*`)`**

cloudera
© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Spark SQL and DataFrames	Distributed Data Processing with Spark

- Spark SQL and the SQL Context
- Creating DataFrames
- Transforming and Querying DataFrames
- Saving DataFrames
- DataFrames and RDDs
- **Comparing Spark SQL, Impala and Hive-on-Spark**
- Conclusion
- Homework: Use Spark SQL for ETL

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Comparing Impala to Spark SQL

- **Spark SQL is built on Spark, a *general purpose* processing engine**
  - Provides convenient SQL-like access to structured data in a Spark application
- **Impala is a *specialized* SQL engine**
  - Much better performance for querying
  - Much more mature than Spark SQL
  - Robust security via Sentry
- **Impala is better for**
  - Interactive queries
  - Data analysis
- **Use Spark SQL for**
  - ETL
  - Access to structured data required by a Spark application

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Comparing Spark SQL with Hive on Spark

- **Spark SQL**
  - Provides the DataFrame API to allow structured data processing *in a Spark application*
  - Programmers can mix SQL with procedural processing
- **Hive-on-Spark**
  - Hive provides a SQL abstraction layer over MapReduce or Spark
    - Allows non-programmers to analyze data using familiar SQL
  - Hive-on-Spark replaces MapReduce as the engine underlying Hive
    - Does not affect the user experience of Hive
    - Except many times faster queries!

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Spark SQL and DataFrames	Distributed Data Processing with Spark

- Spark SQL and the SQL Context
- Creating DataFrames
- Transforming and Querying DataFrames
- Saving DataFrames
- DataFrames and RDDs
- Comparing Spark SQL, Impala and Hive-on-Spark
- **Conclusion**
- Homework: Use Spark SQL for ETL

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Spark SQL is a Spark API for handling structured and semi-structured data**
- **Entry point is a SQLContext**
- **DataFrames are the key unit of data**
- **DataFrames are based on an underlying RDD of Row objects**
- **DataFrames query methods return new DataFrames; similar to RDD transformations**
- **The full Spark API can be used with Spark SQL Data by accessing the underlying RDD**
- **Spark SQL is not a replacement for a database, or a specialized SQL engine like Impala**
  - Spark SQL is most useful for ETL or incorporating structured data into other applications

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Spark SQL and DataFrames	Distributed Data Processing with Spark

- Spark SQL and the SQL Context
- Creating DataFrames
- Transforming and Querying DataFrames
- Saving DataFrames
- DataFrames and RDDs
- Comparing Spark SQL, Impala and Hive-on-Spark
- Conclusion
- **Homework: Use Spark SQL for ETL**

cloudera © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Homework: Use Spark SQL for ETL

- **In this homework assignment you will**
  - Import the data from MySQL
  - Use Spark to normalize the data
  - Save the data to Parquet format
  - Query the data with Impala or Hive
- **Please refer to the Homework description**

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.