• Open Source analytics engine for large-scale data processing
• Provides interface to program entire clusters
• APIs in Scala, Java, Python and R + Interactive Spark-Shell in Scala
**RDDs**

- Resilient distributed dataset
- Immutable distributed collection of data
- Can be cached in memory across the cluster
- Manipulation through parallel functional operators
- Operators can be chained

**Limitations:**
- Datamodel = opaque blobs
- No optimizations possible

**DataFrame API**

- Data is organized into named columns, like a table in a relational database

**Benefits:**
- Declarativity allows query plan optimization
- Strongly typed data model allows for optimized storage

**Limitations:**
- No custom lambdas possible, first have to be converted to RDDs
- Syntax checking is limited

**Dataset API**

- Combination of RDD and DataFrames

**Benefits:**
- Object-oriented programming interface
- Optionally also weakly typed objects are allowed
- When only strong typed used - everything can be checked during compile time

**DataFrame = Dataset [Row]**
Expressive queries using SQL

Streaming data analysis instead of only batch analysis

Provides machine learning algorithms

An API tailored towards analyzing graphs and also implementing custom graph algorithms

Spark Core
Task distribution, scheduling, I/O functionalities

Spark SQL
Expressive queries using SQL

Spark Streaming
Streaming data analysis instead of only batch analysis

MLib
Provides machine learning algorithms

GraphX
An API tailored towards analyzing graphs and also implementing custom graph algorithms
Initialization

Transformations

Actions

- CSV
- select()
- filter()
- join()
- union()
- sort()
- limit()

- show()
- count()
```scala
val studenten = spark.read.format("csv").schema(StructType(
  List(
    StructField("matrnr", IntegerType, false),
    StructField("name", StringType, false),
    StructField("semester", IntegerType, false)
  )
)).option("delimiter", ",").load("studenten.csv")
```

<table>
<thead>
<tr>
<th>matrnr</th>
<th>name</th>
<th>semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>24002</td>
<td>Xenokrates</td>
<td>18</td>
</tr>
<tr>
<td>25403</td>
<td>Jonas</td>
<td>12</td>
</tr>
<tr>
<td>26120</td>
<td>Fichte</td>
<td>10</td>
</tr>
<tr>
<td>26830</td>
<td>Aristoxenos</td>
<td>8</td>
</tr>
<tr>
<td>27550</td>
<td>Schopenhauer</td>
<td>6</td>
</tr>
<tr>
<td>28106</td>
<td>Carnap</td>
<td>3</td>
</tr>
<tr>
<td>29120</td>
<td>Theophrastos</td>
<td>2</td>
</tr>
<tr>
<td>29555</td>
<td>Feuerbach</td>
<td>2</td>
</tr>
</tbody>
</table>
- DataFrame API

Transformation

- `select()`
- `filter() / where()`
- `join()`
- `union()`, `intersect()`, `except()`
- `sort() / orderBy()`
- `limit()`
- `groupBy() + agg()`
- DataFrame API - Column

• Represents a column in a Dataset that holds a Catalyst Expression that produces a value per row.

• How to generate Column references:
  
  • With a $-prefixed string: "$matrnr"
  
  • With the "col" or "column" functions: col("matrnr")
  
  • From a dataset: studenten("matrnr")

• With column references as base types, more complex expression trees can be build:

  • when("semester" <= 3, "Grundstudium").otherwise("Hauptstudium")

  • "$semester" === 18 && "$name".startsWith("X")
DataFrame API

- **select()**
- **filter() / where()**
- **join()**
- **union(), intersect(), except()**
- **sort() / orderBy()**
- **limit()**
- **groupBy() + agg()**

```
studenten.select("matrnr", "name")
studenten.filter("name" === "Fichte")
studenten.join(hoeren, hoeren("matrnr") === studenten("matrnr"))
val studierende = studenten.union(studentinnen)
studenten.sort("matrnr".desc)
studenten.limit(3)
```

```
vorlesungen
  .groupBy("gelesenvon")
  .agg(count("*").as("#vorlesungen"),
       sum("sws").as("gesamtstunden"))
```
- DataFrame API - join

• Inner join

```scala
studenten.join(hoeren, hoeren("matrnr") === studenten("matrnr"))
```

• Specify join type as third argument

```scala
studenten.join(hoeren, hoeren("matrnr") === studenten("matrnr"), "leftsemi")
```

• Supported types: 'inner', 'fullouter', 'leftouter', 'rightouter', 'leftsemi', 'leftanti', 'cross'

• Self-Join

```scala
studenten.as("a").join(studenten.as("b"), "$a.matrnr" === "$b.matrnr")
```
- **DataFrame API**

**Actions**

- show()
  
  ```scala
  studenten.show(8)
  ```

  shows top 20 rows when no parameter is passed

- count()
  
  ```scala
  val anzahlStudenten = studenten.count()
  ```

---

<table>
<thead>
<tr>
<th>matrnr</th>
<th>name</th>
<th>semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>24002</td>
<td>Xenokrates</td>
<td>18</td>
</tr>
<tr>
<td>25403</td>
<td>Jonas</td>
<td>12</td>
</tr>
<tr>
<td>26120</td>
<td>Fichte</td>
<td>10</td>
</tr>
<tr>
<td>26830</td>
<td>Aristoxenos</td>
<td>8</td>
</tr>
<tr>
<td>27550</td>
<td>Schopenhauer</td>
<td>6</td>
</tr>
<tr>
<td>28106</td>
<td>Carnap</td>
<td>3</td>
</tr>
<tr>
<td>29120</td>
<td>Theophrastos</td>
<td>2</td>
</tr>
<tr>
<td>29555</td>
<td>Feuerbach</td>
<td>2</td>
</tr>
</tbody>
</table>
References

- Foundations in Data Engineering (Lecture 4): Distributed Processing
- https://spark.apache.org/
- http://www.tpc.org/tpch/
Hands-on session
TPC-H Benchmark

- **Transaction Processing Performance Council (TPC)**
  - Big influence on the industry standard benchmarks
  - Companies use TPC-Benchmarks to demonstrate their competitiveness
  - The TPC committee belongs to large database vendors like IBM, Microsoft, Oracle and HP

- **TPC-H** is a decision-support benchmark. It consists of ad-hoc queries and concurrent data modifications
  - The Database schema is in third normal form and contains **8 tables**
  - The Benchmark can be executed on different sizes of data. This can be configured with the scale factor. **Scale factor 1 corresponds to 1 GB of data**
    - 6 of the 8 tables grow linearly with the scale factor
  - There exist **22 complex queries** as well as two INSERT and UPDATE processes which are executed in parallel to test concurrency
  - The official specification how to execute the TPC-H benchmark is 137 pages long
  - On the website new results are published and the official specification can be downloaded: [tpc.org](http://tpc.org)
Preparations

• Start Spark in Scala Shell:
  • Navigate to your spark directory
  • Start ./bin/spark-shell

• Add necessary imports:
  • import org.apache.spark.sql.types._
  • import org.apache.spark.sql.
Exercise 1

• Load the region.tbl data into a data frame
Load all tables

- Load the tpch.scala file into your Spark Shell:
  - Download **tpch.scala**: [https://tinyurl.com/3383aa4n](https://tinyurl.com/3383aa4n)
    - `wget https://tinyurl.com/3383aa4n/download/tpch.scala`
  - Update **DATA_PATH** variable in **tpch.scala**:
    ```scala
def DATA_PATH = /the/path/to/your/tpc-h/data
```
  - Load the script into your shell:
    - Option 1: Load the script into running Spark Shell:
      ```scala
      :load /path/to/tpch.scala
      ```
    - Option 2: Restart Spark:
      ```bash
      ./$SPARK_HOME/bin/spark-shell -I /path/to/tpch.scala
      ```
Exercise 2

• Show only the name of the regions
Exercise 3

• Count the nations that are not located in Europe
Exercise 4

• Which was the biggest order in 1996?
Exercise 5

• Which customer in Europe spent the most money in 1996?
Exercise 6

• Which suppliers have no customers in Europe?
Thank you for your attention