



APACHE  
**Spark**<sup>TM</sup>



- 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

# APACHE Spark™ - Programming Model

## 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]**

# APACHE Spark™ - Stack

## 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

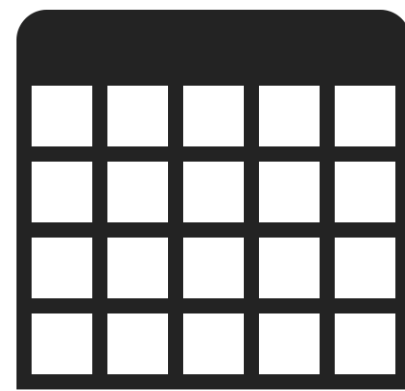
Provides machine  
learning algorithms

## GraphX

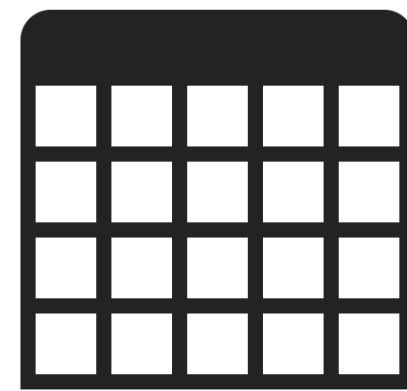
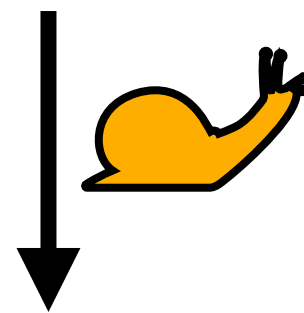
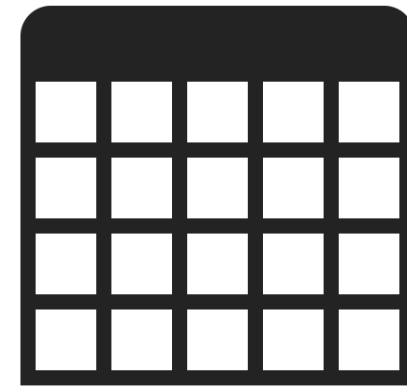
An API tailored towards  
analyzing graphs and  
also implementing  
custom graph  
algorithms

# DataFrame API

## Initialization

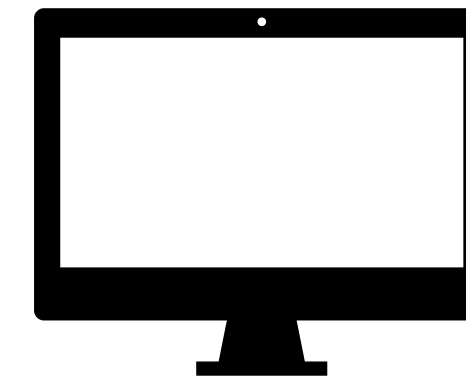
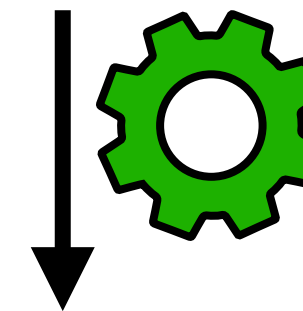
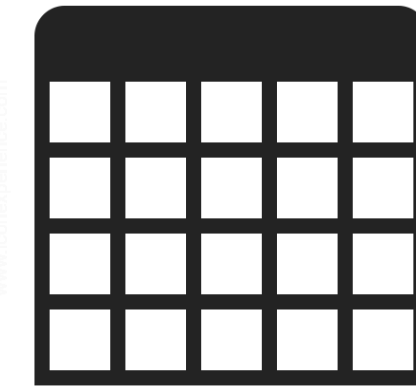


## Transformations



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

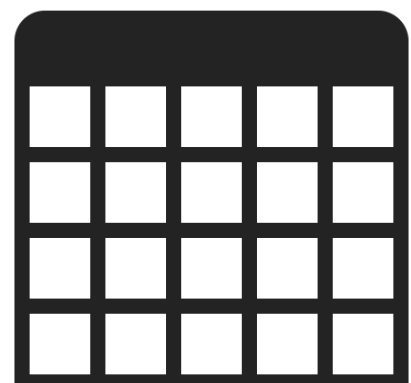
## Actions



- show()
- count()

# APACHE Spark™ - DataFrame API

## Initialization

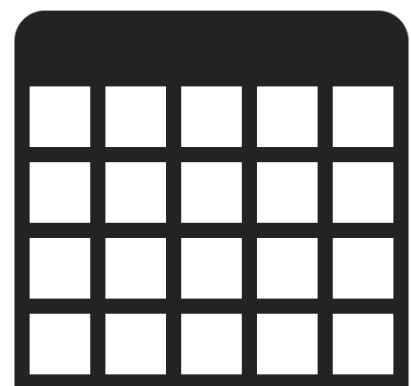
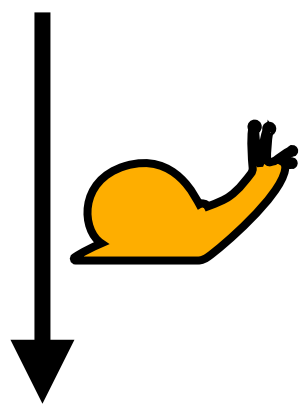
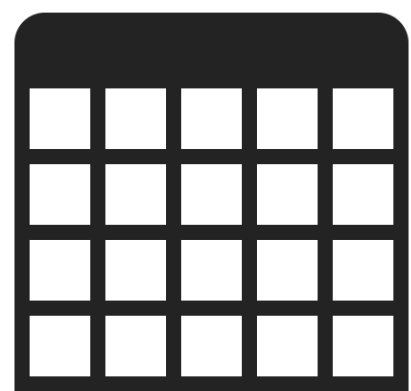


studenten.csv

```
24002|Xenokrates|18  
25403|Jonas|12  
26120|Fichte|10  
26830|Aristoxenos|8  
27550|Schopenhauer|6  
28106|Carnap|3  
29120|Theophrastos|2  
29555|Feuerbach|2
```

```
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")
```

# APACHE **Spark**™ - DataFrame API Transformation

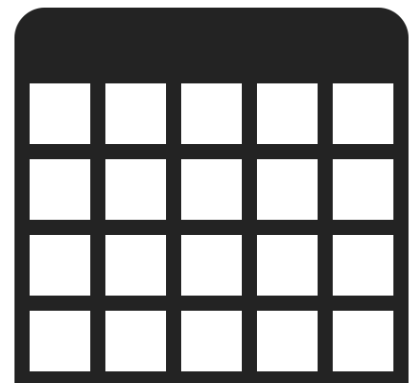
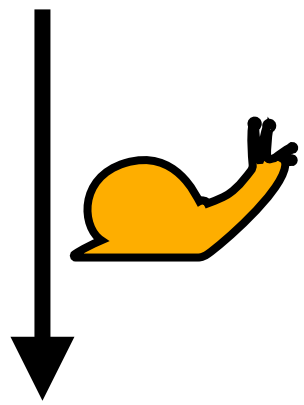
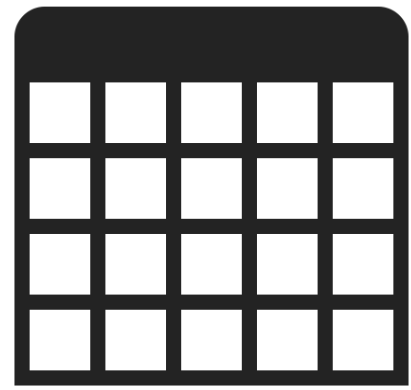


- `select()`
- `filter()` / `where()`
- `join()`
- `union()`, `intersect()`, `except()`
- `sort()` / `orderBy()`
- `limit()`
- `groupBy()` + `agg()`

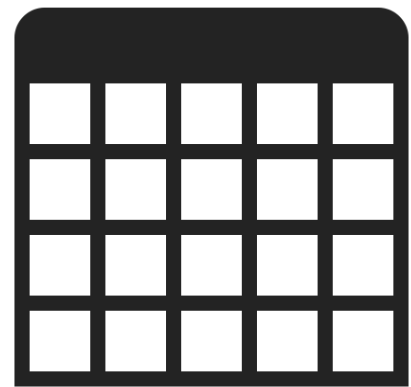
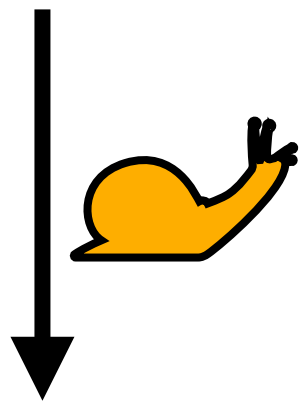
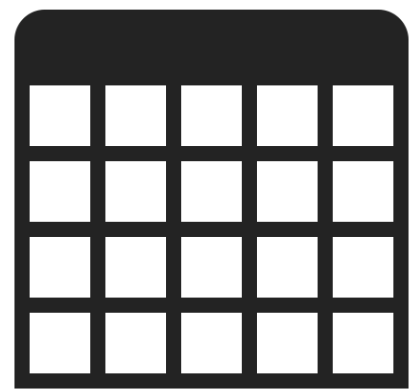


# DataFrame API - Column

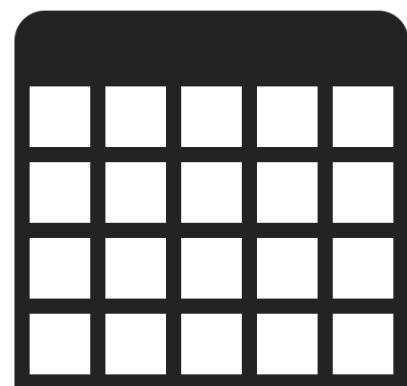
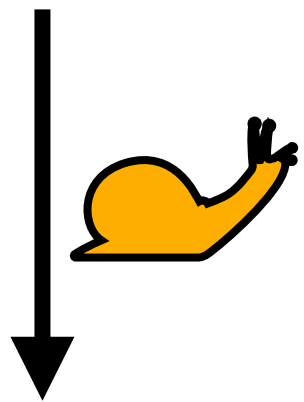
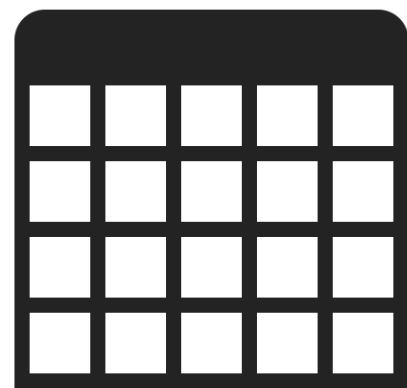
- Represents a column in a Dataset that holds a Catalyst Expression that produces a value per row.







- Represents a column in a Dataset that holds a Catalyst Expression that produces a value per row.
- How to generate Column references:
  - With a `$`-prefixed string:  `$"mat rnr"`
  - With the `"col"` or `"column"` functions:  `col("mat rnr")`
  - From a dataset:  `studenten("mat rnr")`
- With column references as base types, more complex expression trees can be build:
  - `when($"semester" <= 3, "Grundstudium").otherwise("Hauptstudium")`
  - `$"semester" === 18 && $"name".startsWith("X")`



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

```
studenten.select($"matrnr", $"name")
```

```
studenten.filter($"name" === "Fichte")
```

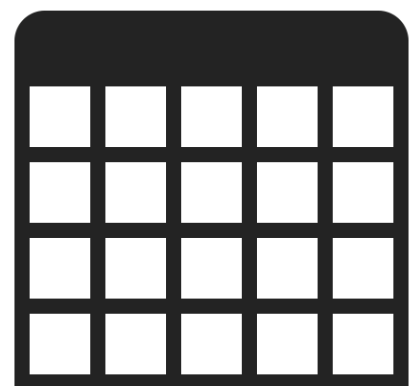
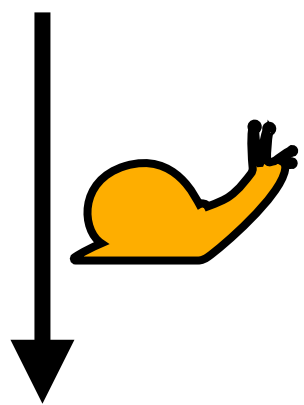
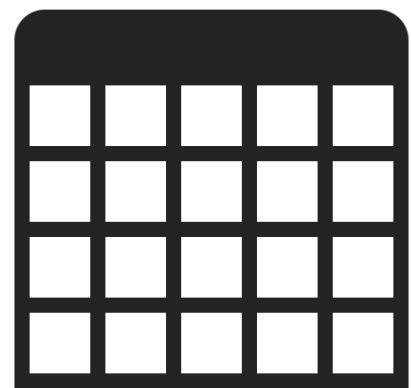
```
studenten.join(hoeren, hoeren("matrnr") === studenten("matrnr"))
```

```
val studierende = studenten.union(studentinnen)
```

```
studenten.sort($"matrnr".desc)
```

```
studenten.limit(3)
```

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



- Inner join

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

- Specify join type as third argument

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

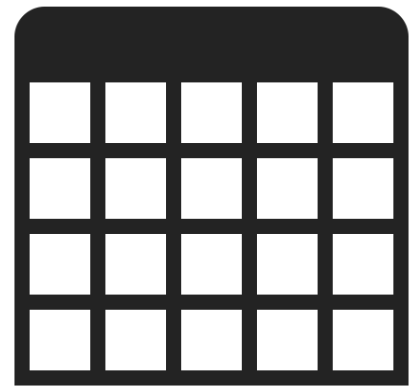
- Supported types: *'inner'*, *'fullouter'*, *'leftouter'*, *'rightouter'*, *'leftsemi'*, *'leftanti'*, *'cross'*

- Self-Join

```
studenten.as("a").join(studenten.as("b"), $"a.matrnr" === $"b.matrnr")
```

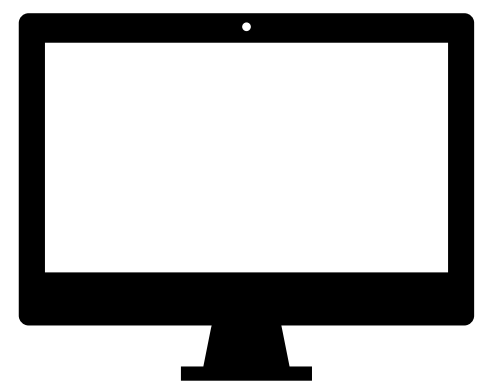
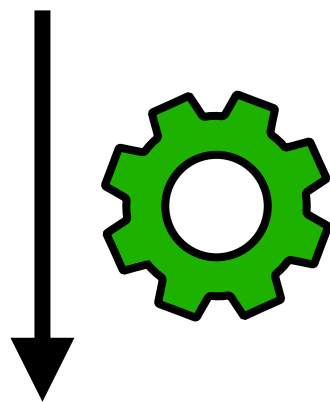
# APACHE Spark™ - DataFrame API

Actions



- `show()` `studenten.show(8)`
  - shows top 20 rows when no parameter is passed

matrn	name	semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2



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

# References

- Foundations in Data Engineering (Lecture 4): Distributed Processing
- <https://spark.apache.org/>
- <https://databricks.com/de/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html>
- <http://www.tpc.org/tpch/>
- <https://www.exasol.com/de/ressource/10-fragen-zum-tpc-h-benchmark/>



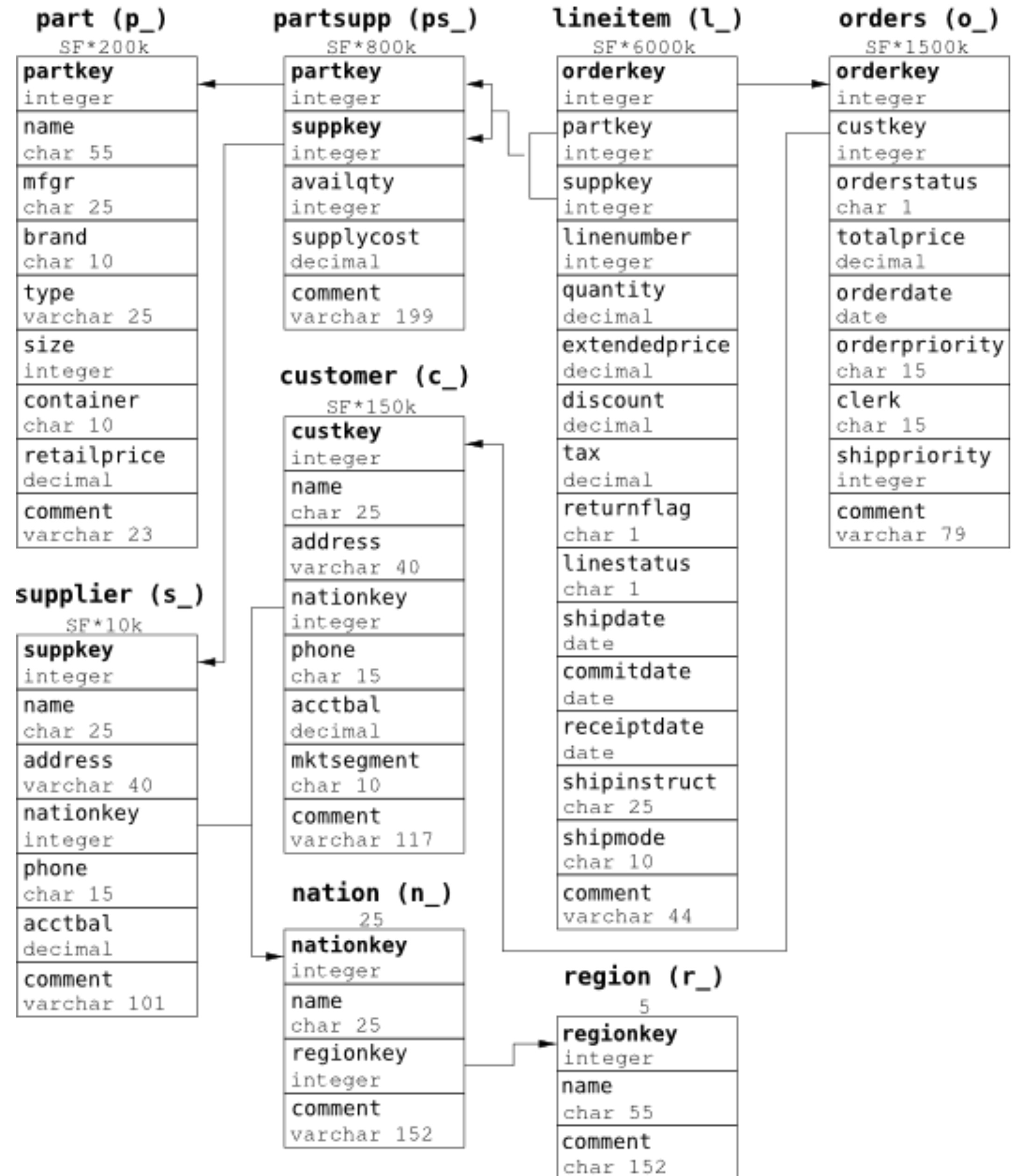
APACHE  
**Spark**<sup>TM</sup>

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)

# TPC-H Schema





# Preparations

- Start Spark in Scala Shell:
  - Navigate to your spark directory
  - Start `./bin/spark-shell --driver-memory 2g`
- 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>  

```
wget https://tinyurl.com/3383aa4n/download/tpch.scala
```
  - **Update `DATA_PATH` variable in `tpch.scala`:**  

```
val DATA_PATH = /the/path/to/your/tpc-h/data
```
  - **Load the script into your running Spark Shell:**  

```
:load /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 4

- Which was the biggest order in 1996?



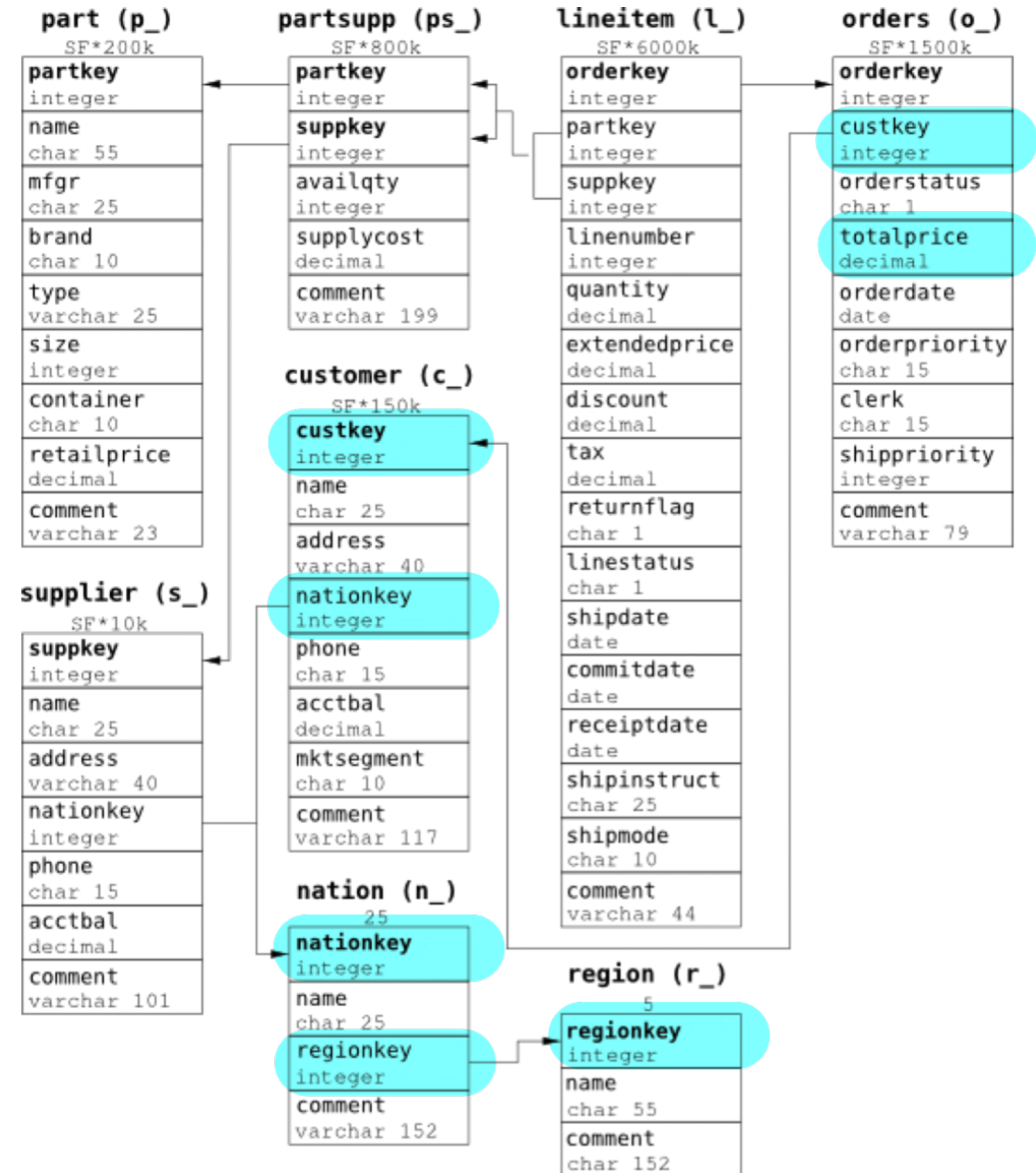
# Exercise 5

- Which customer in Europe spent the most money in 1996?



# Exercise 5

- Which customer in Europe spent the most money in 1996?



# Exercise 6

- Which suppliers have no customers in Europe?

# Exercise 6

- Which suppliers have no customers in Europe?

