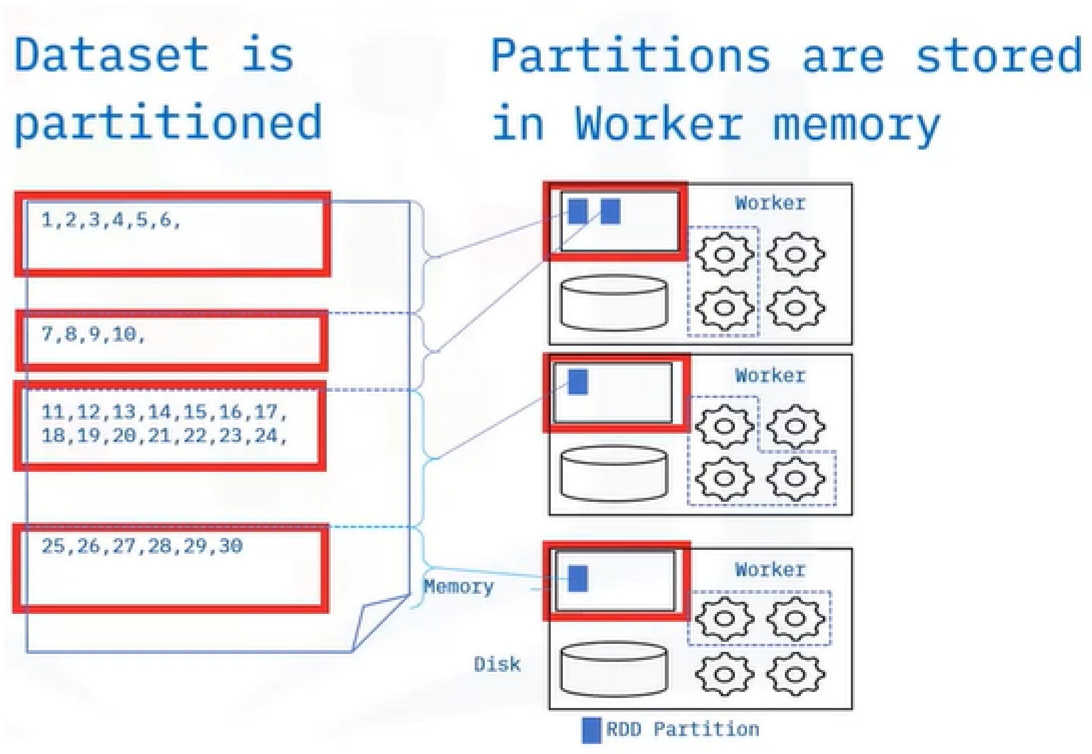
Week 4: Introduction to DataFrames and SparkSQL

RDDs in Parallel Programming and Spark

Resilient Distributed Datasets

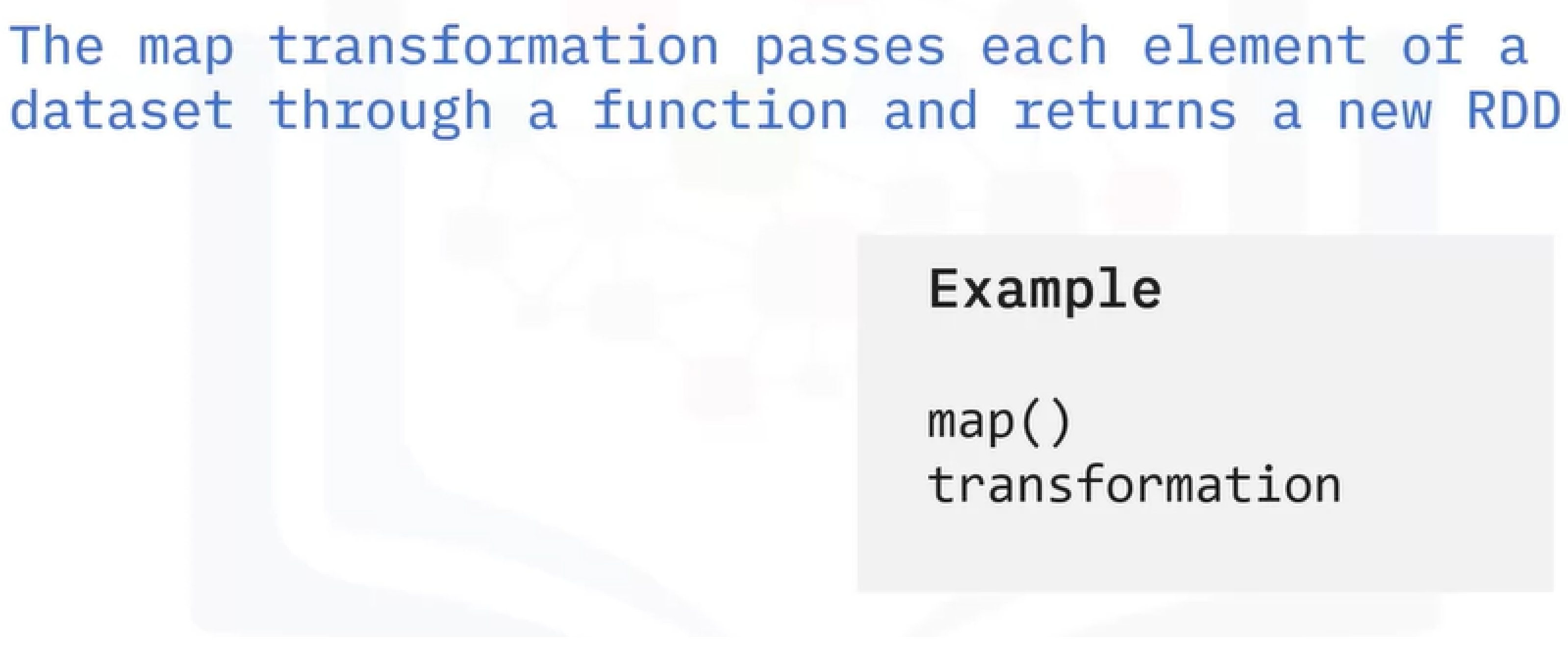
Resilient Distributed Datasets, known as RDDs

- Are Spark's primary data abstraction
- Are partitioned across a cluster's nodes.



RDD transformation

- Creates a new RDD from an existing RDD.
- Transformations in Spark are considered lazy because Spark does not compute transformation results immediately. Instead, the results are only computed when evaluated by "actions."



The map transformation passes each element of a dataset through a function and returns a new RDD

Example

map() transformation

RDD action

To evaluate a transformation in Spark, you use an action.

The action returns a value to the driver program after running a computation.

💡 One example is the reduce action, which aggregates all the elements of an RDD and returns the result to the driver program.

Directed Acyclic Graph, knowns as a DAG

But how do transformations and actions happen?

Spark uses a unique data structure called a Directed Acyclic Graph, knowns as a DAG, and an associated DAG Scheduler to perform RDD operations.

- A graphical structure composed of edges and vertices.
- The term "acyclic" means that new edges only originate from an existing vertex. In general, the vertices and edges are sequential.
- The vertices represent RDDs
  The edges represent the transformations or actions.

The DAG Scheduler applies the graphical structure to run the tasks that use the RDD to perform transformation processes.

DAGS help enable fault tolerance. When a node goes down, Spark replicates the DAG and restores the node.

Transformation and Action

1. Spark creates DAG when creating an RDD.
2. Spark enables the DAG Schedular to perform a transformation and updates the DAG.
3. The DAG now points to the new RDD.
4. The pointer that transforms RDD is returned to the Spark driver program.
5. If there is an action, the driver program that calls the action evaluates the DAG only after Spark completes the action.

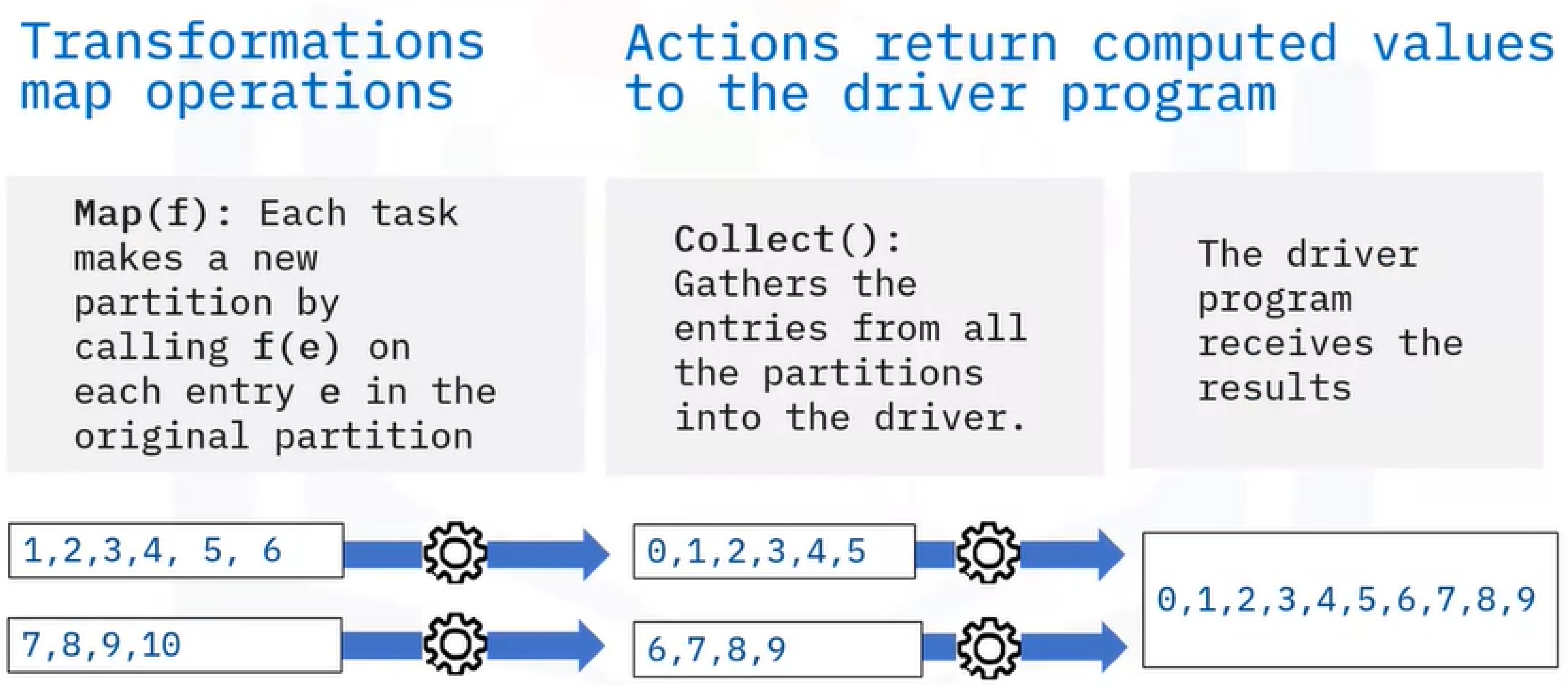
Transformation examples

map *(func)*	Returns a new distributed dataset formed by passing each element of the source throught a function *func*
filter *(func)*	Returns a new dataset formed by selecting those
distinct ([numTasks])	Returns a new dataset that contains the distinct elements of the source dataset
flatma *(func)*	Similar to map *(func)* Can map each input item to zero or more output items *Func* should return a Seq rather than a single item

Action examples

reduce *(func)* aggregate dataset elements using the function *func*	*func* takes 2 arguments and returns one: - Is commutative - Is associative - Can be computed correctly in parallel
take(n)	Returns an array with the first *n* element
coleect()	Returns all the elements as an array WARNING: Make sure that ? will fit in driver program
takeOrdered(n,key=func)	Returns n elements ordered in ascending order or as specified by the optional key function

A transformation with an action


Transformations map operations
Actions return computed values to the driver program
Map(f): Each task makes a new partition by calling f(e) on each entry e in the original partition
Collect(): Gathers the entries from all the partitions into the driver.
The driver program receives the results
1,2,3,4, 5, 6
7,8,9,10
0,1,2,3,4,5
6,7,8,9
0,1,2,3,4,5,6,7,8,9

A transformation is only a mapping of operations.

You need actions to get the computed values to the driver program:

This example considers a function f of x that decrements x by 1. So, f of x equals x minus 1. You'll apply this function as a transformation to a dataset using the map transformation. Each task makes a new partition by calling f of e on each entry e in the original partition. Then, the collect action gathers the entries from all the partitions into the driver, which receives the results.

Conclusion

- RDDs are Spark's primary data abstraction partitioned across the nodes of the cluster.
- Spark uses DAGS to enable fault tolerance.
- When a node goes down, Spark replicates the DAG and restores the node.
- Transformations leave existing RDDs intact and create new RDDs based on the transformation function.
- Transformations undergo lazy evaluation, meaning they are only evaluated when the driver function calls an action.

Data-frames and Datasets

Datasets

Datasets are the newest Spark data abstraction.

Datasets are a collection of strongly typed Java Virtual Machine, or JVM, objects.

Strongly typed means that datasets are typesafe, and the dataset's datatype is made explicit during its creation.

Datasets provide the benefits of both RDDs (such as lambda functions, type-safety), and SparkSQL (SQL Optimizations)

Datasets features

Datasets are immutable, data cannot be deleted or lost.

Datasets feature an encoder that converts type-specified JVM objects to a tabular representation.

Datasets extend the DataFrame API.

Because datasets are strongly typed, APIs are currently only available in Scala and Java, which are statically typed languages.

Dynamically typed languages, such as Python and R, do not support dataset APIs.

Datasets in Spark - benefits

Because Datasets are statically-typed, **datasets provide compile-time type safety.**

Compile-time type safety means that Spark can detect syntax and semantic errors in production applications before deployment, saving substantial developer and operational costs and time.

Datasets compute faster than RDDs, especially for aggregate queries.

Datasets offer the additional query optimization enabled by Catalyst and Tungsten.

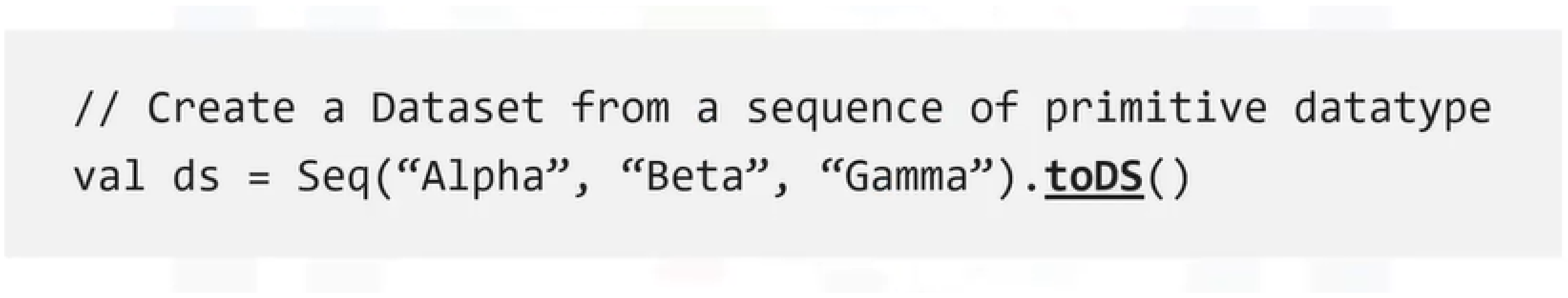
Datasets enable improved memory usage and caching.

The dataset API also offers functions for convenient high-level aggregate operations, including sum, average, join, and "group-by."

Create a Datasets

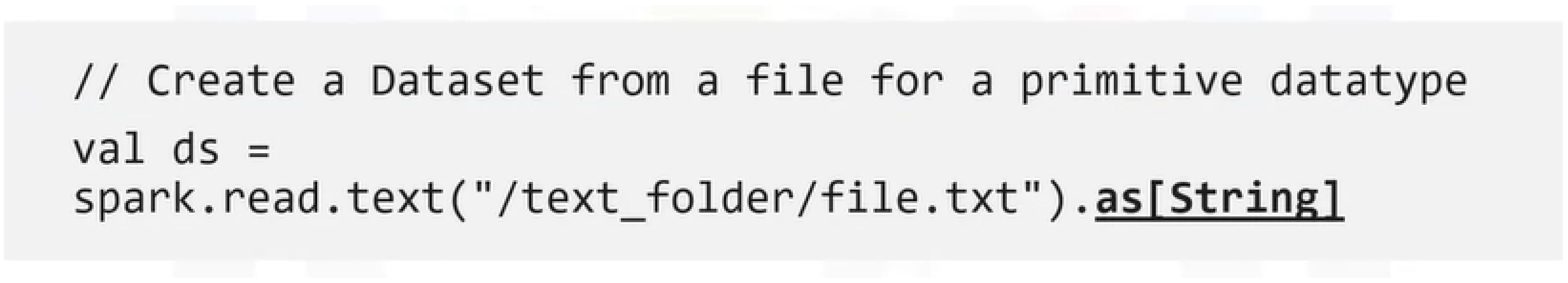
Three ways that you can create a dataset within Spark.

`#1 Create datasetsfrom a sequence` - written in Scala, uses the toDS function



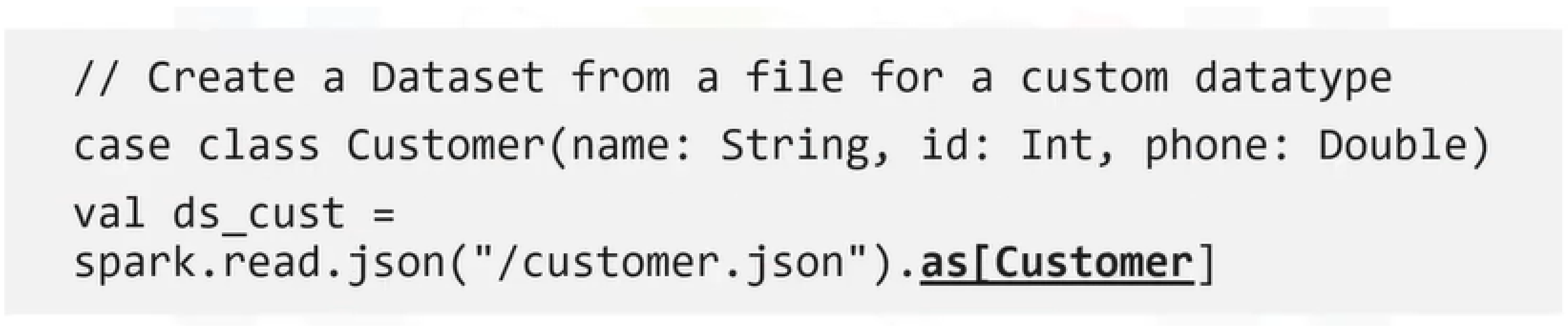
```
// Create a Dataset from a sequence of primitive datatype
val ds = Seq("Alpha", "Beta", "Gamma").toDS()
```

`#2 Create datasetsfrom a text file` - explicit schema declaration



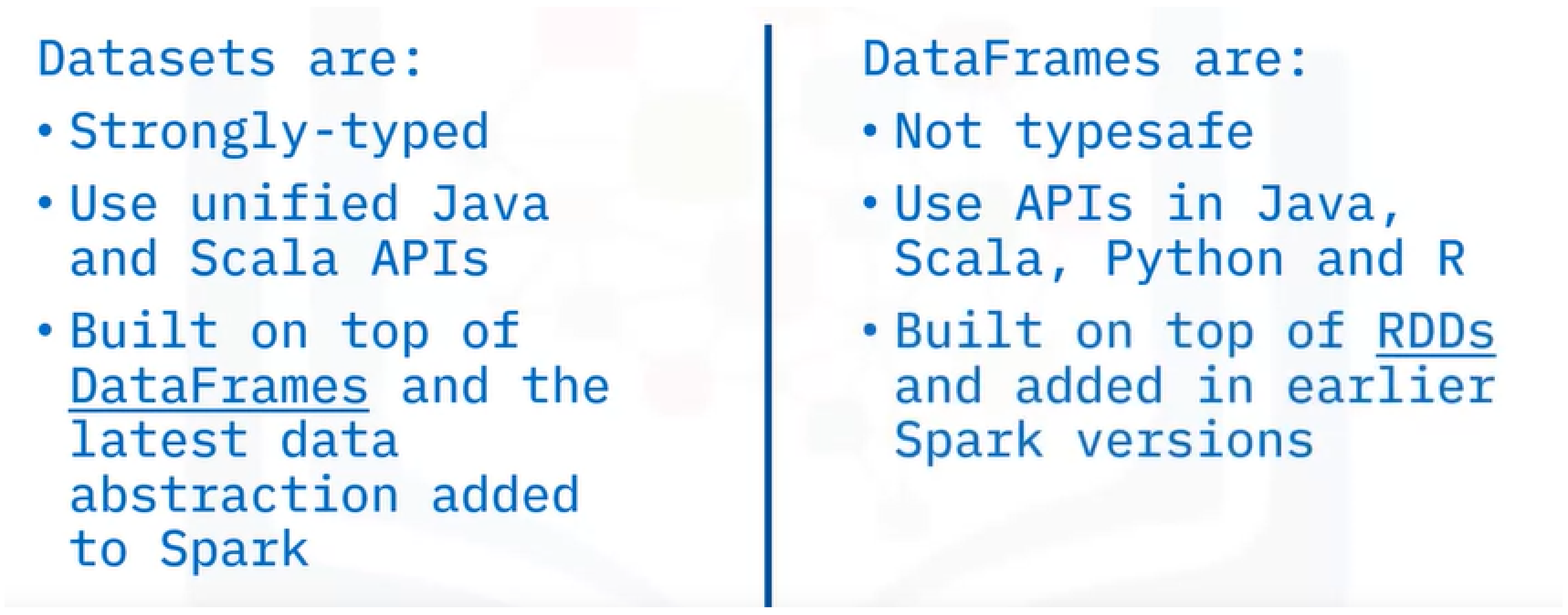
```
// Create a Dataset from a file for a primitive datatype
val ds =
spark.read.text("/text_folder/file.txt").as[String]
```

`#3 Create datasetsfrom using JSON file` - use a custom class called "Customer," which contains a "name" and an "ID" field.



```
// Create a Dataset from a file for a custom datatype
case class Customer(name: String, id: Int, phone: Double)
val ds_cust =
spark.read.json("/customer.json").as[Customer]
```

Datasets & DataFrames



Datasets are:
- Strongly-typed
- Use unified Java and Scala APIs
- Built on top of DataFrames and the latest data abstraction added to Spark

DataFrames are:
- Not typesafe
- Use APIs in Java, Scala, Python and R
- Built on top of RDDs and added in earlier Spark versions

Datasets offer the benefits of both its predecessors – DataFrames and RDDs.

Datasets	DataFrames
Strongly typed	Not typesafe
Use unified Java and Scala APIs	Use APIs in Java, Scala, Python, and R

The latest addition to Spark and are built on top of DataFrames	Introduced earlier, are built on RDDs

Conclusion

- A dataset is a distributed collection of data that provides the combined benefits of both RDDs and SparkSQL.
- Datasets consist of strongly typed JVM objects.
- Datasets use DataFrame typesafe capabilities and extend object-oriented API capabilities.
- Datasets work with both Scala and Java APIs.

Spark SQL Memory Optimization using Catalyst and Tungsten

Spark SQL optimization goals

- Improve the SQL query run-time performance
- Reduce the query's time
- Reduce memory consumption
- Saving organizations time and money

Catalyst defined

Spark SQL supports both rule-based and cost-based query optimization.

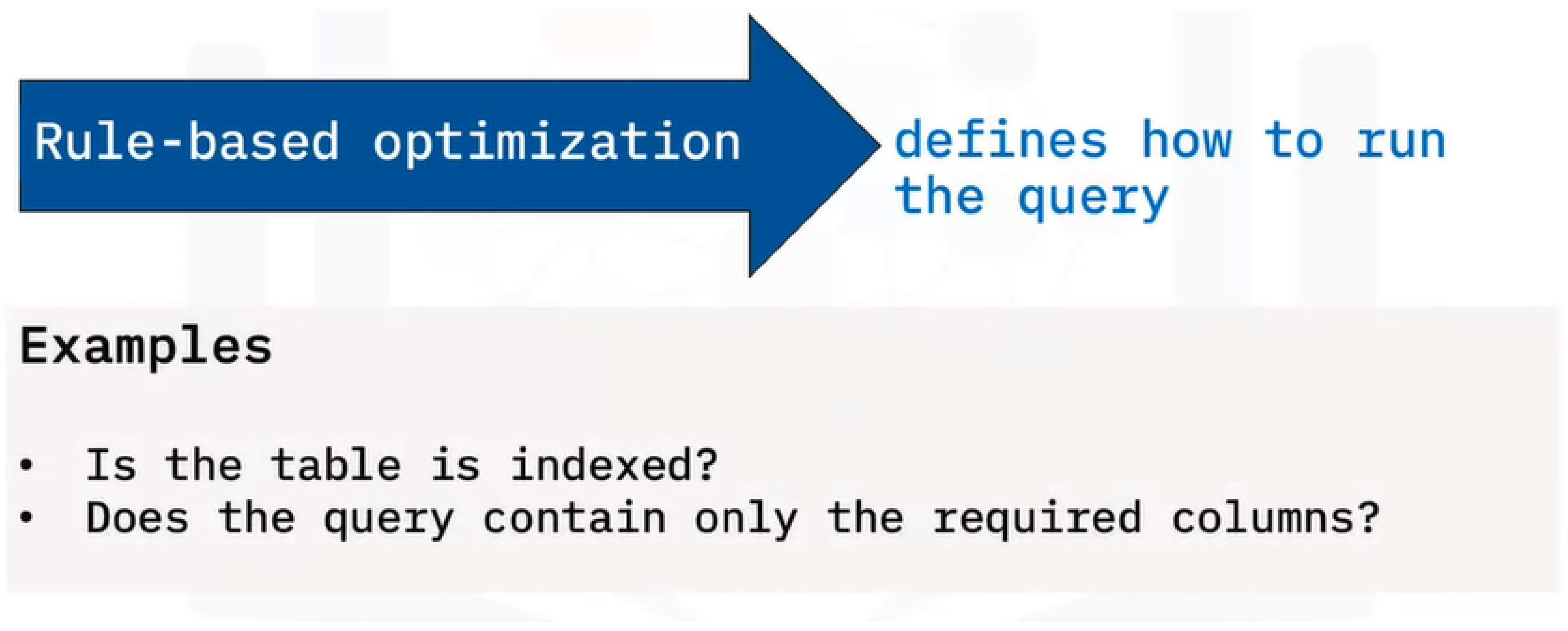
Catalyst, also known as the Catalyst Optimizer:

- Is the Spark SQL built-in rule-based query optimizer.
- Based on Scala functional programming constructs,
- Catalyst is designed to easily add new optimization techniques and features to Spark SQL.
- Developers can extend the optimizer by adding data-source-specific rules and support for new data types.

SQL optimization explained

Rule-based

During rule-based optimization, the SQL optimizer follows predefined rules that determine how to run the SQL query.

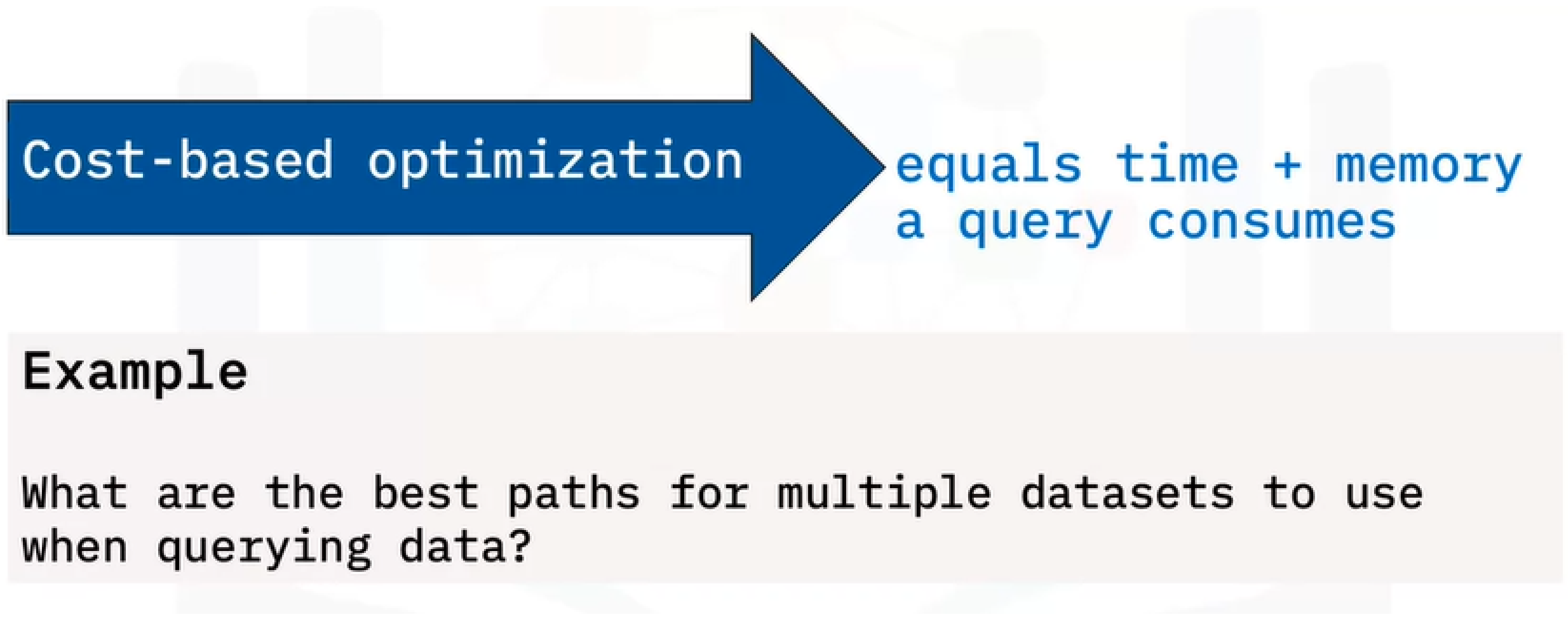


Rule-based optimization defines how to run the query

Examples

- Is the table is indexed?
- Does the query contain only the required columns?

Cost-based


Cost-based optimization

equals time + memory a query consumes

Example

What are the best paths for multiple datasets to use when querying data?

With the query itself optimized, cost-based optimization measures and calculates cost based on the time and memory that the query consumes.

💡 The Catalyst optimizer selects the query path that results in the lowest time and memory consumption. Because queries can use multiple paths, these calculations can become quite complex when large datasets are part of the calculation.

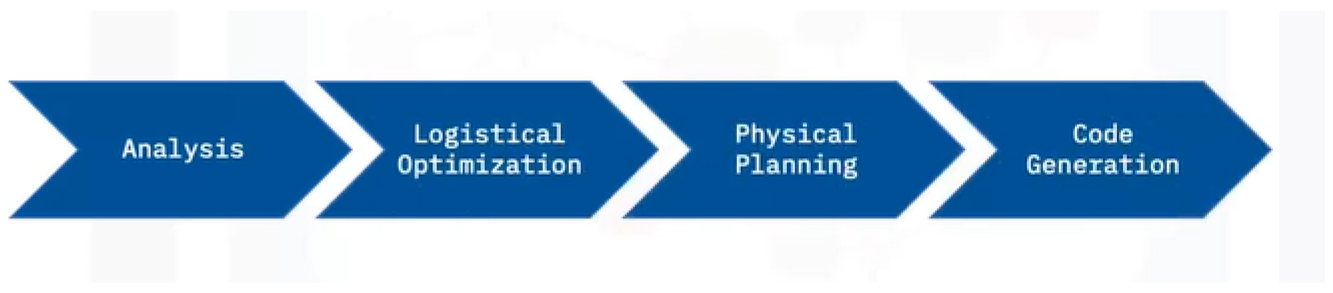
Example

A car owner can optimize the vehicle's performance with the right oil, tires, fuel, and so on.
These actions are similar to rule-based optimization.

When it's time to plan a travel route that can save both time and wear and tear on your car, those actions are a form of cost-based optimization.

Catalyst query optimization

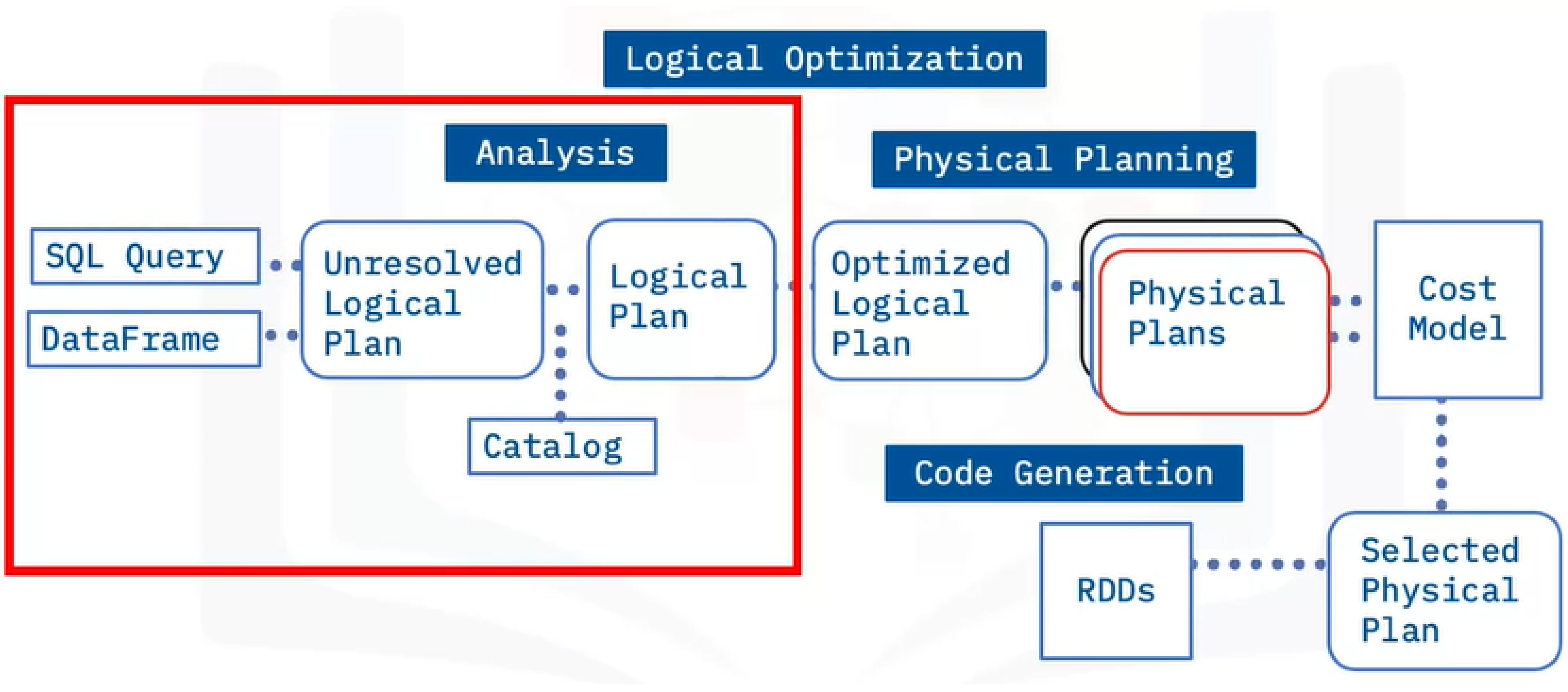
- Uses a tree data structure and provides the data tree rule sets.
- To optimize a query, Catalyst performs the following four high-level tasks or phases
  - analysis
  - logical optimization
  - physical planning
  - code generation.



Review the four Catalyst phases

Analysis phase

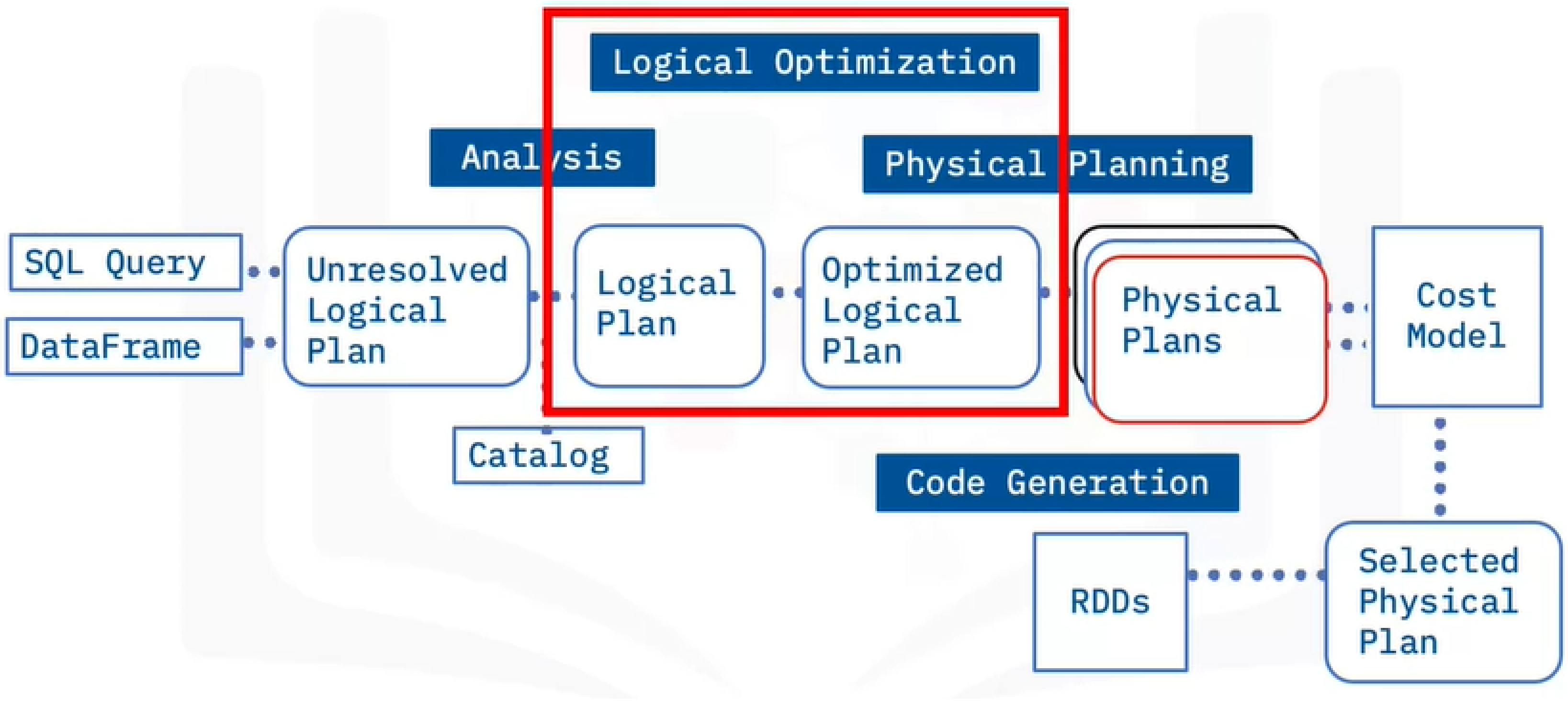
Catalyst analyzes the query, the DataFrame, the unresolved logical plan, and the Catalog to create a logical plan.



Logical Optimization phase

During the Logical Optimization phase, the Logical plan evolves into an Optimized Logical Plan.

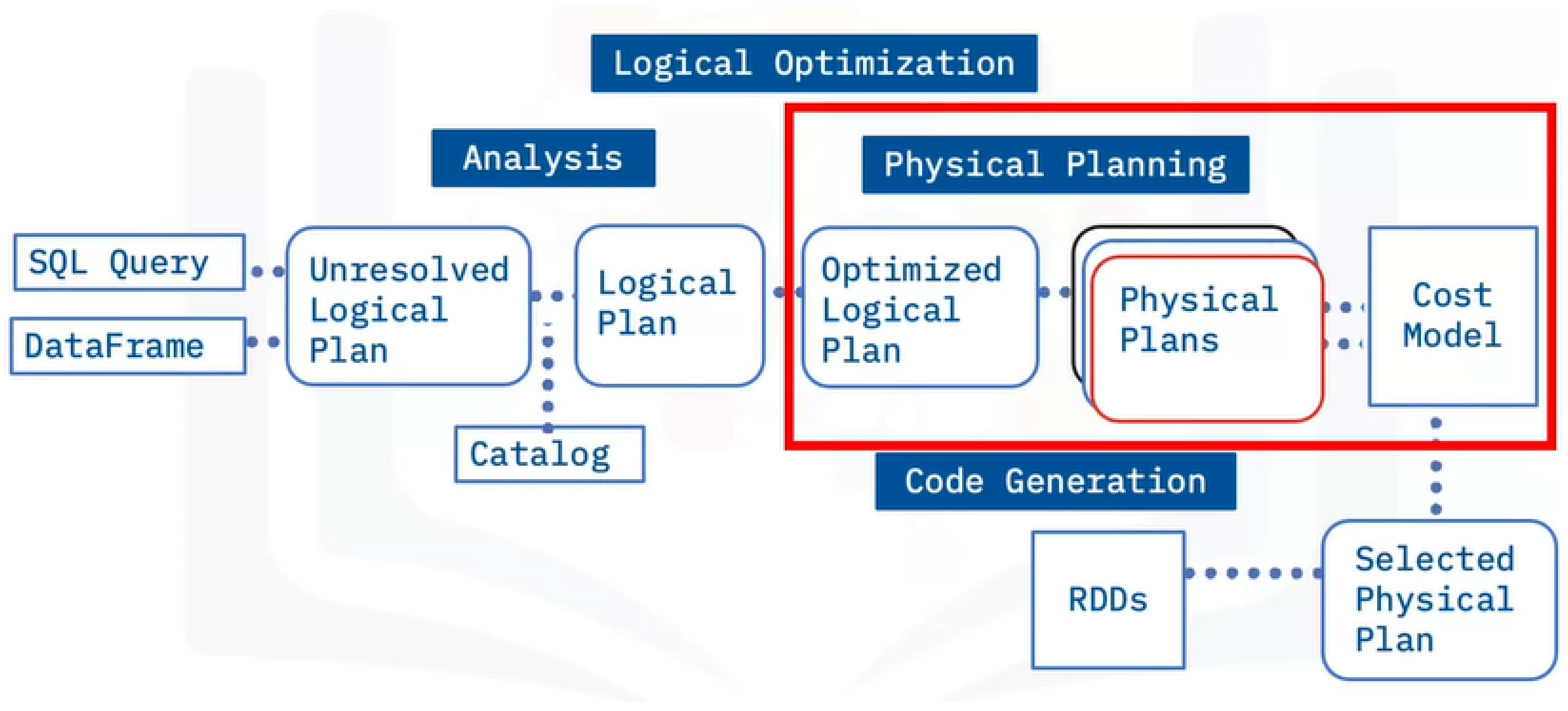
This is the **rule-based optimization** step of Spark SQL and rules such as *folding, pushdown, and pruning* are applied here.



Physical Planning phase

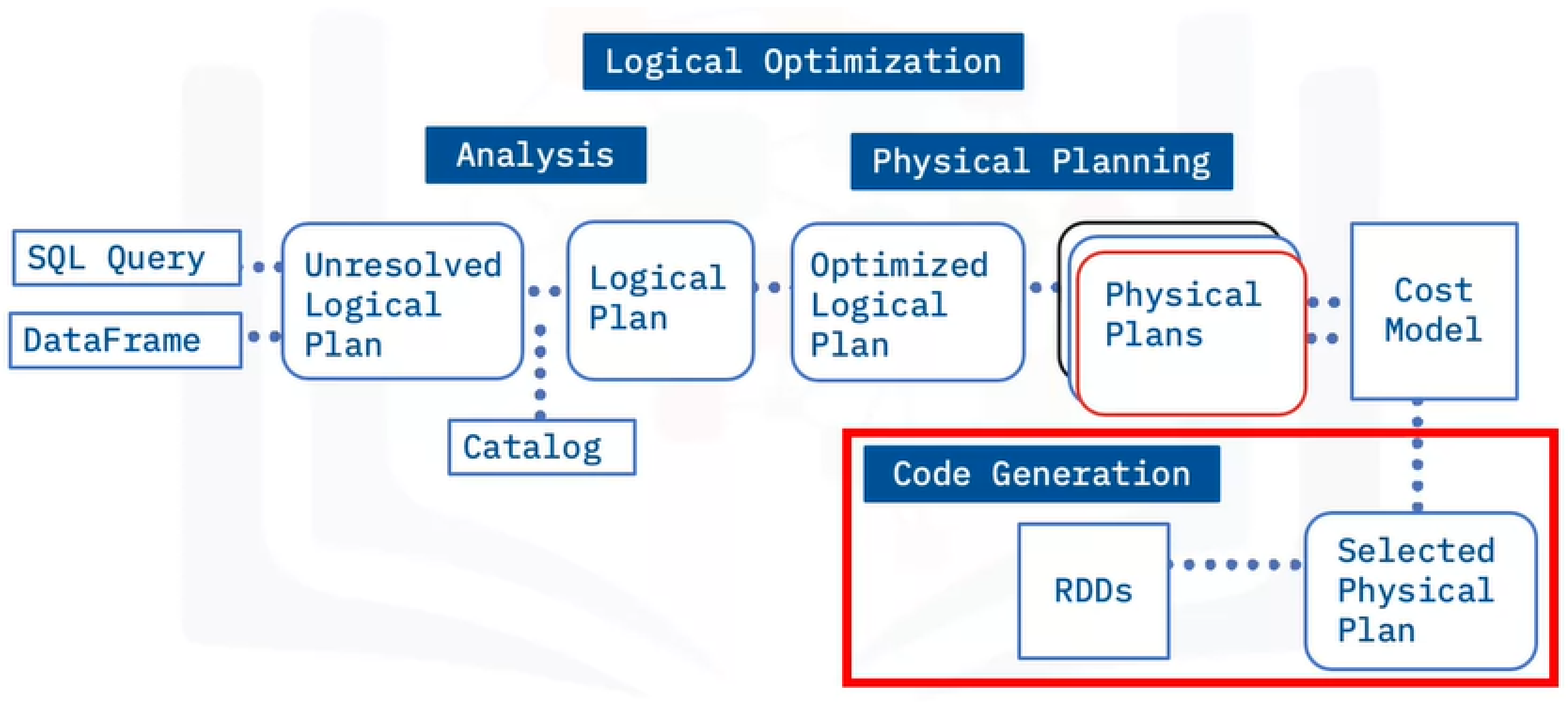
Catalyst generates multiple physical plans based on the Logical Plan.

A physical plan describes computation on datasets with specific definitions on how to conduct the computation. A cost model then chooses the physical plan with the least cost. This is the **cost-based optimization** step.



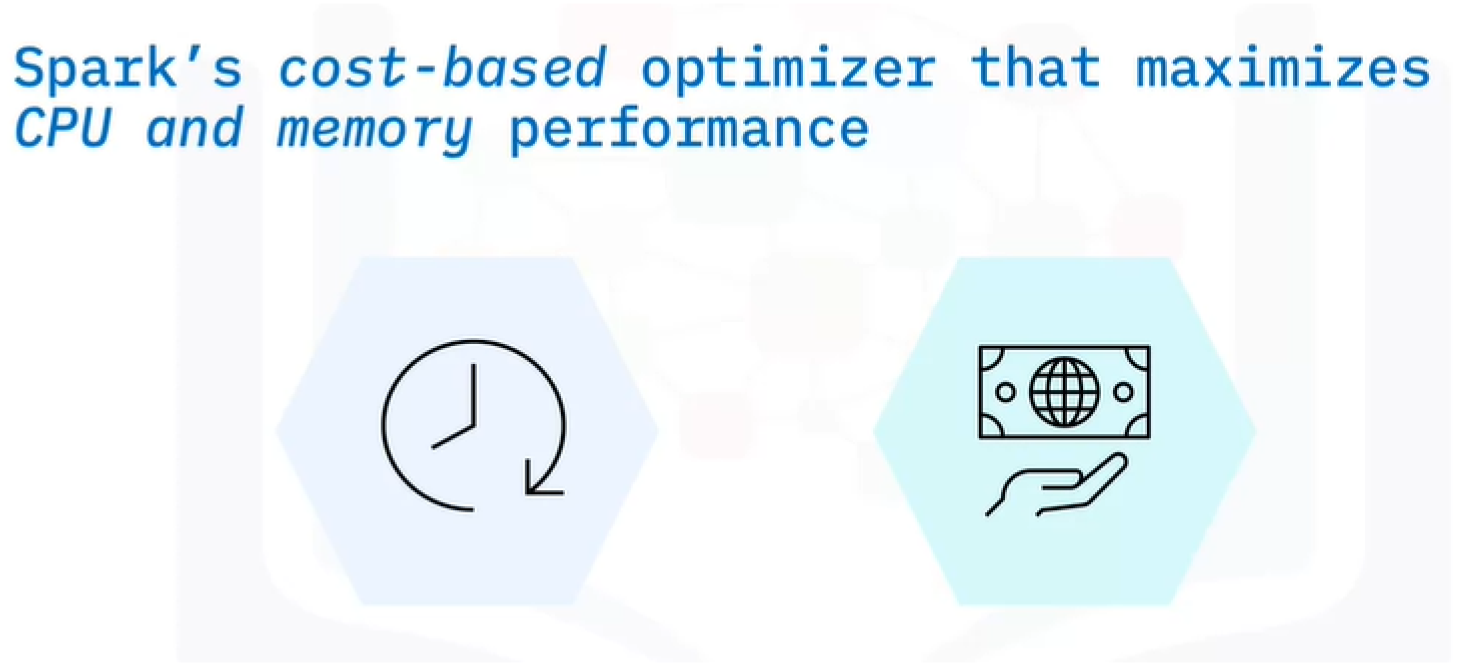
Code Generation

Catalyst applies the selected physical plan and generates Java bytecode to run on each node.



Tungsten defined

Tungsten provides cost-based optimization in Spark.


Spark's cost-based optimizer that maximizes CPU and memory performance

Tungsten optimizes the performance of underlying hardware focusing on **CPU performance instead of IO**.

Java was initially designed for transactional applications. Tungsten aims to improve CPU and Memory performance by using a method more suited to data processing for the JVM to process data.

Tungsten features

- Tungsten manages memory explicitly and does not rely on the JVM object model or garbage collection.
- Tungsten creates cache-friendly data structures that are arranged easily and more securely using STRIDE-based memory access instead of random memory access.
- Tungsten supports JVM bytecode on-demand.
- Tungsten does not enable virtual function dispatches, reducing multiple CPU calls.
- Tungsten places intermediate data in CPU registers
- Enables loop unrolling.

Conclusion

- Catalyst is the Spark SQL built-in rule-based query optimizer.
- Catalyst performs analysis, logical optimization, physical planning, and code generation.
- Tungsten is the Spark built-in cost-based optimizer for CPU and memory usage that enables cache-friendly computation of algorithms and data structures.

ETL with DataFrames

Basic DataFrame operations

`Reading, Analysis, Transformation, Loading, and Writing.`

Spark:

- **Reads in the data**, loads the data into a DataFrame.
- **Analyze the dataset**
  starting with simple tasks such as examining the columns, data types, number of rows, and optionally working with aggregated stats, trend analysis, and other operations.
- **Transform the data**
  filtering the data to locate specific values, sorting the dataset based on specific criteria, or by joining this dataset with another dataset.
- **Load transformed dataset** back to a database or a file system
- **Write the data** back to disk.

`This process is commonly also referred to as ETL – Extract, Transform, Load.`

ETL is an essential process in any data processing pipeline as the first step makes data accessible to data warehouses for machine learning models, downstream applications, or other services.

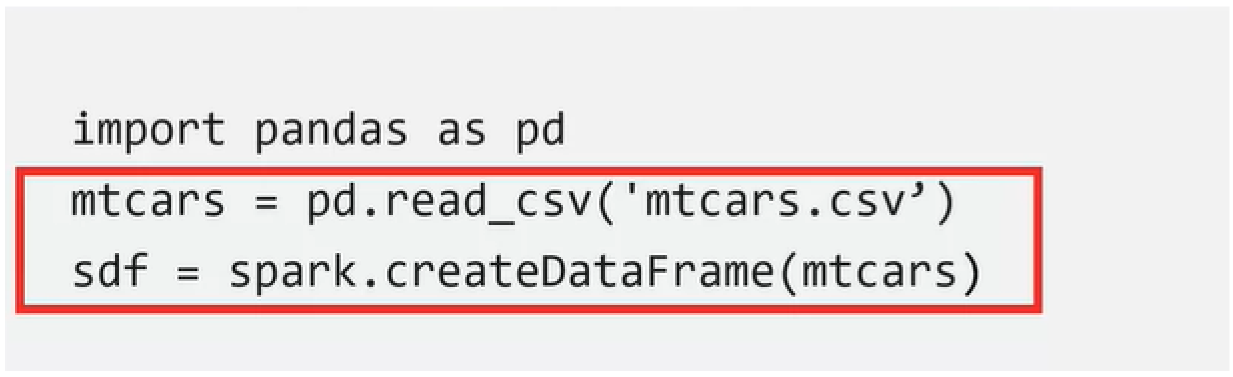
💡 Another standard process. ELT – Extract, Load, and Transform. This process emerged as a result of big data processing.

With ELT, all of the data resides in a **data lake**. A data lake is a vast pool of raw data, for which the purpose of the data is not pre-defined. When a data lake exists, each project forms individual transformation tasks as required, in contrast to anticipating all necessary transformation requirements usage scenarios as done with ETL and a data warehouse. Organizations often use a mixture between both ETL and ELT.

Read the data

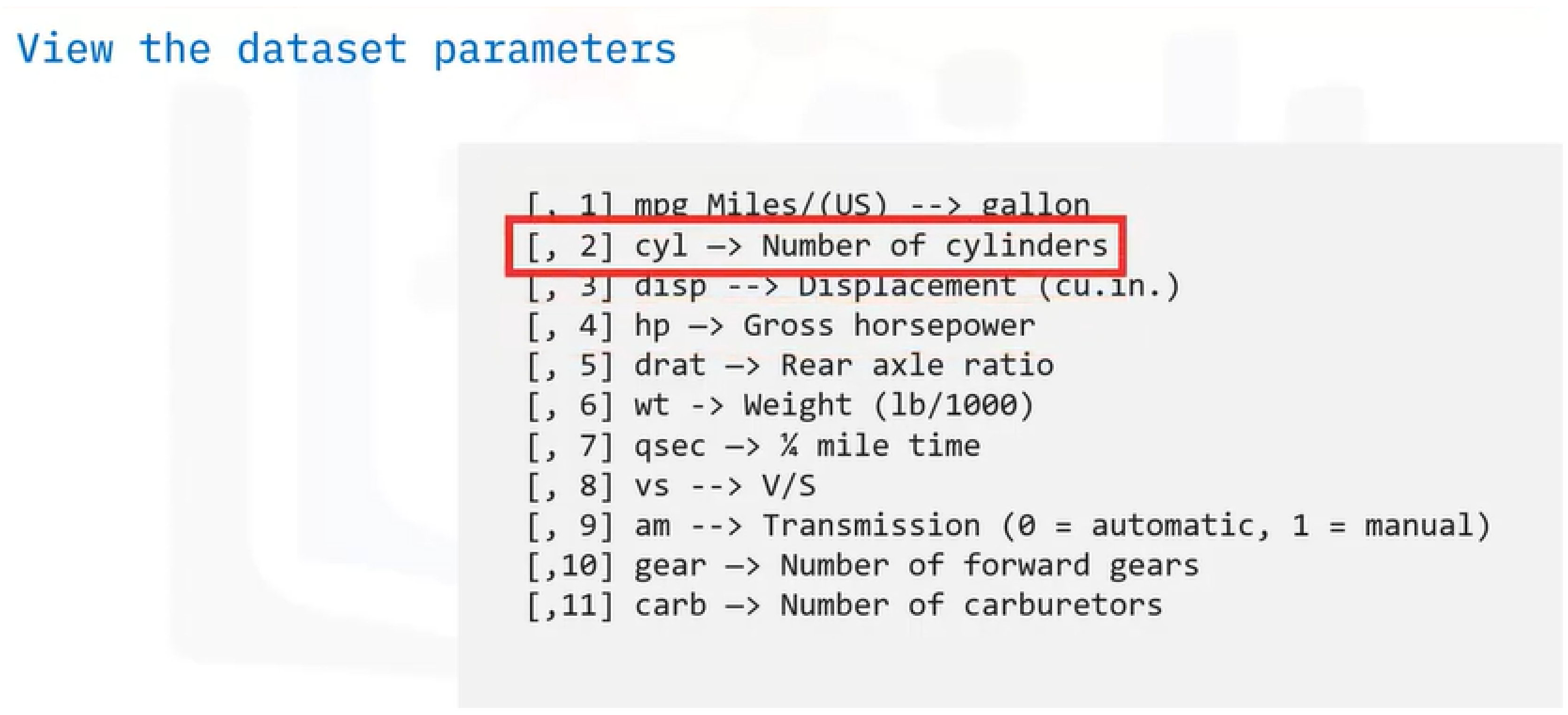
Can directly load data into DataFrames (or)

Create a new Spark DataFrame from an existing DataFrame.



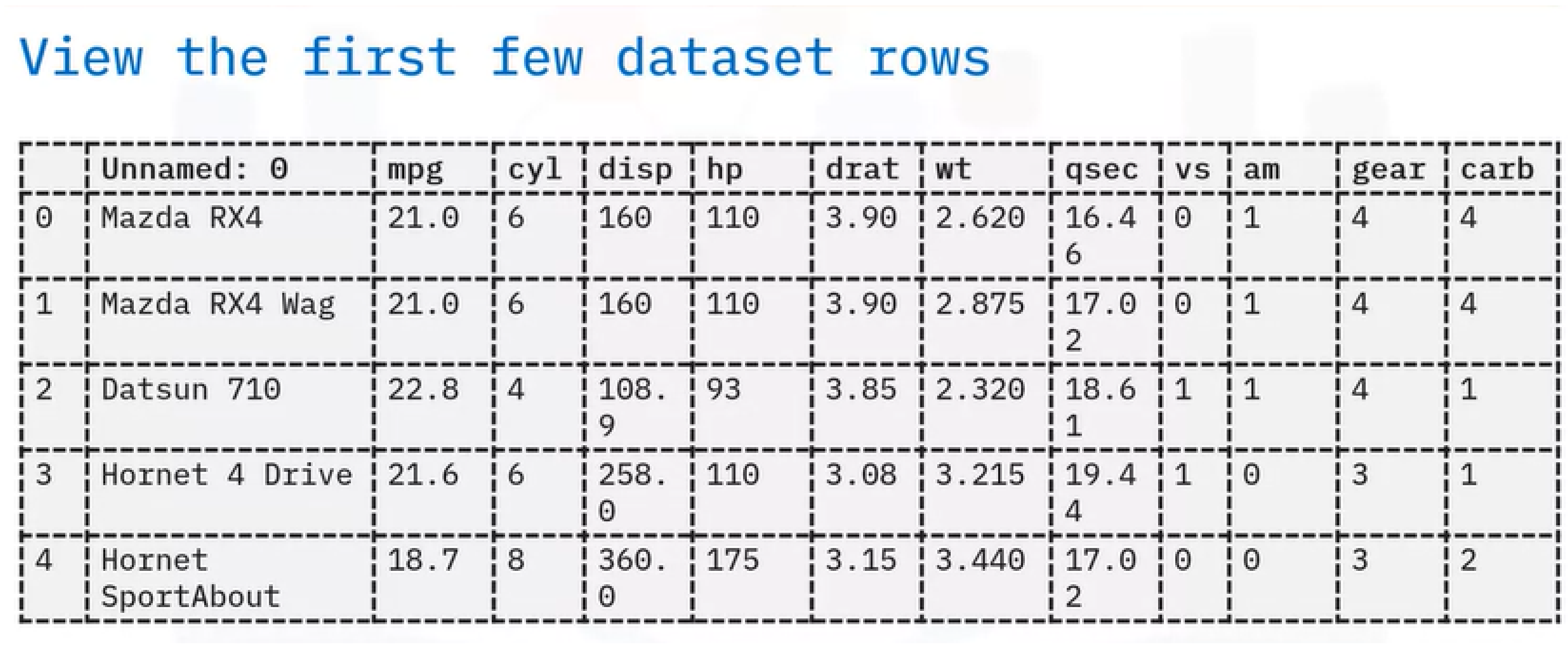
```
import pandas as pd
mtcars = pd.read_csv('mtcars.csv’)
sdf = spark.createDataFrame(mtcars)
```

The displayed code sample shows how to load a dataset into a Pandas DataFrame in Python and then load that same dataset into a Spark DataFrame object.



View the dataset parameters

```
[, 1] mpg Miles/(US) --> gallon
[, 2] cyl –> Number of cylinders
[, 3] disp --> Displacement (cu.in.)
[, 4] hp –> Gross horsepower
[, 5] drat –> Rear axle ratio
[, 6] wt -> Weight (lb/1000)
[, 7] qsec –> ¼ mile time
[, 8] vs --> V/S
[, 9] am --> Transmission (0 = automatic, 1 = manual)
[,10] gear –> Number of forward gears
[,11] carb –> Number of carburetors
```



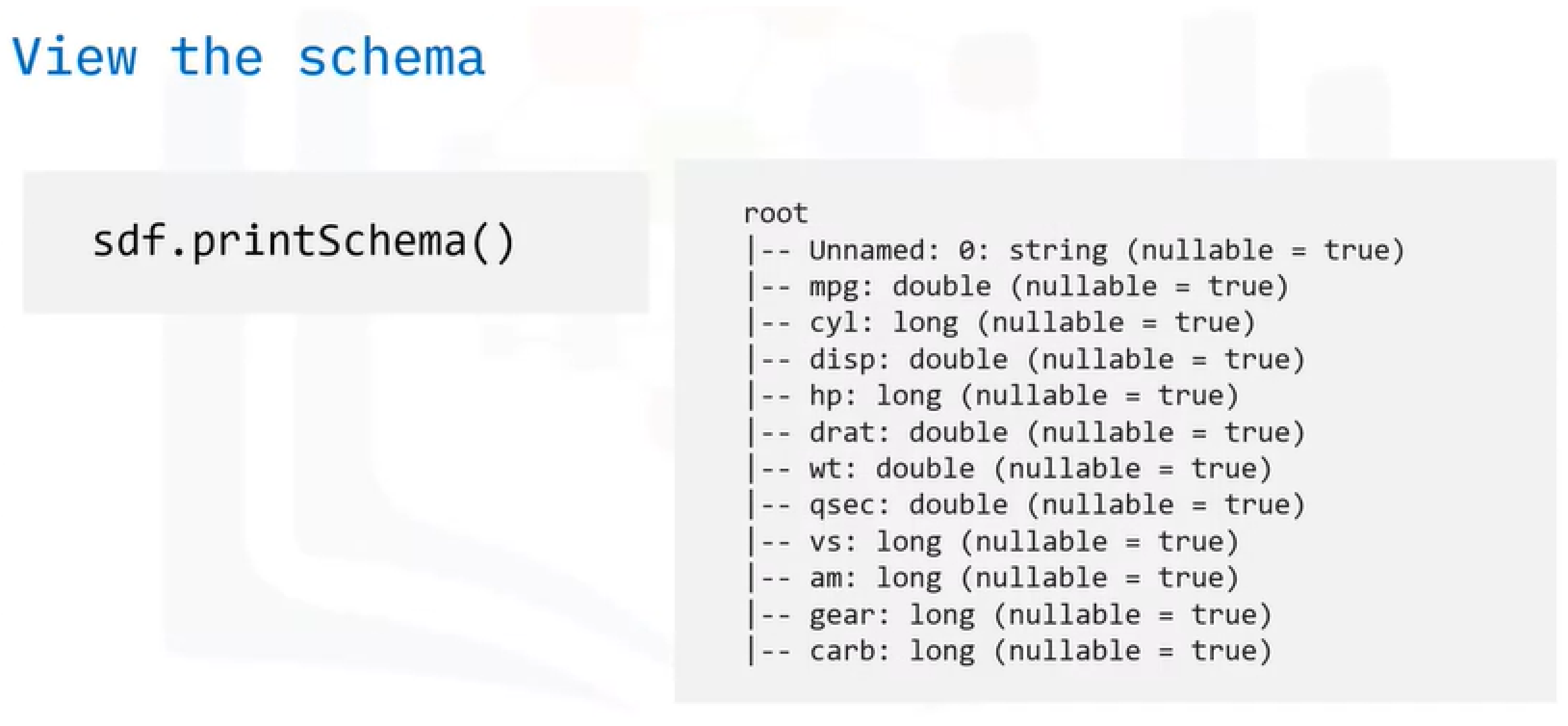
View the first few dataset rows

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.9	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.6	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet SportAbout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Analyze the data

Using the printschema

Examine the DataFrame column data types or the schema, using the print schema method. Take note of each column's data types.



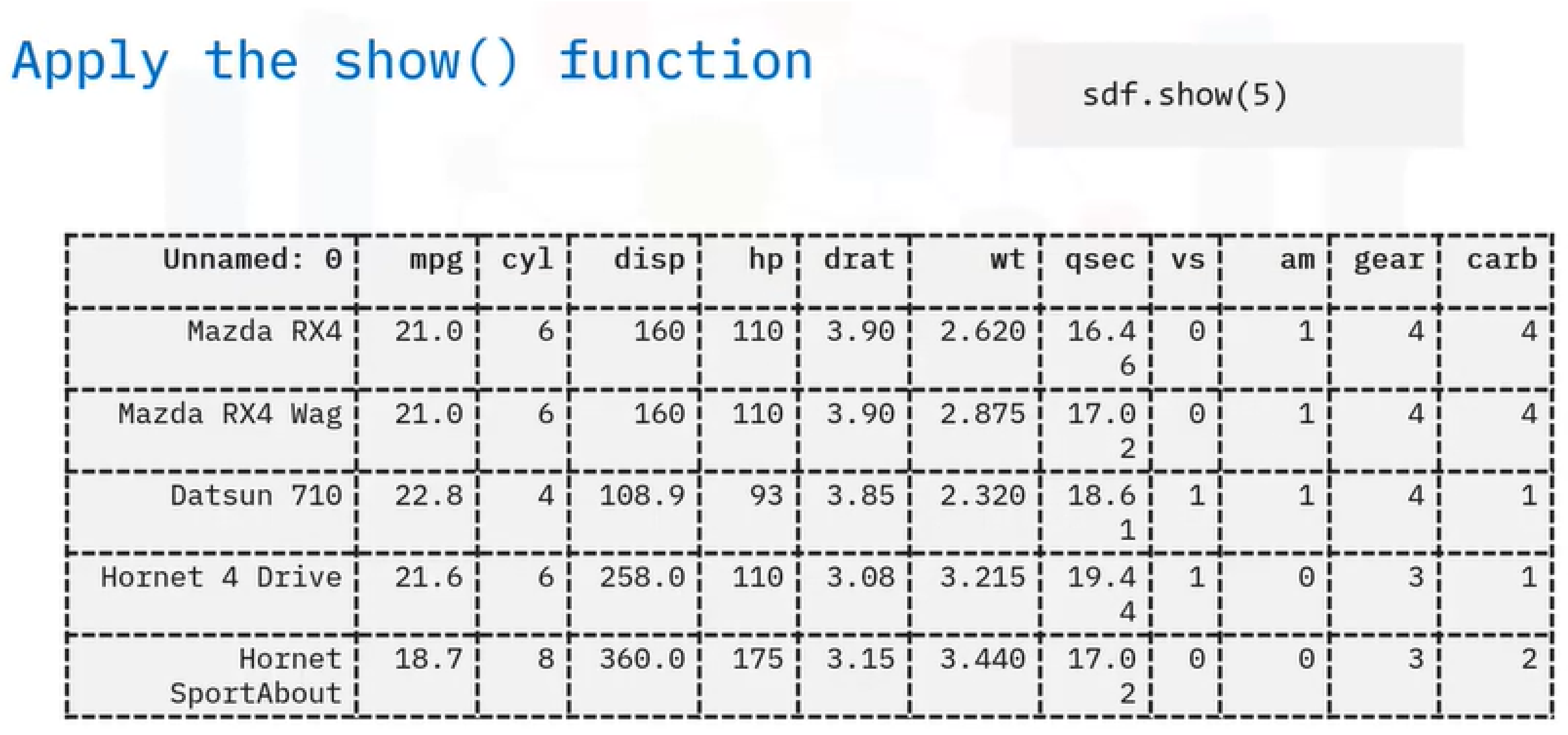
View the schema

```
sdf.printSchema()
```

```
root
 |-- Unnamed: 0: string (nullable = true)
 |-- mpg: double (nullable = true)
 |-- cyl: long (nullable = true)
 |-- disp: double (nullable = true)
 |-- hp: long (nullable = true)
 |-- drat: double (nullable = true)
 |-- wt: double (nullable = true)
 |-- qsec: double (nullable = true)
 |-- vs: long (nullable = true)
 |-- am: long (nullable = true)
 |-- gear: long (nullable = true)
 |-- carb: long (nullable = true)
```

Using the show function

examine a specified number of DataFrame using the show function.



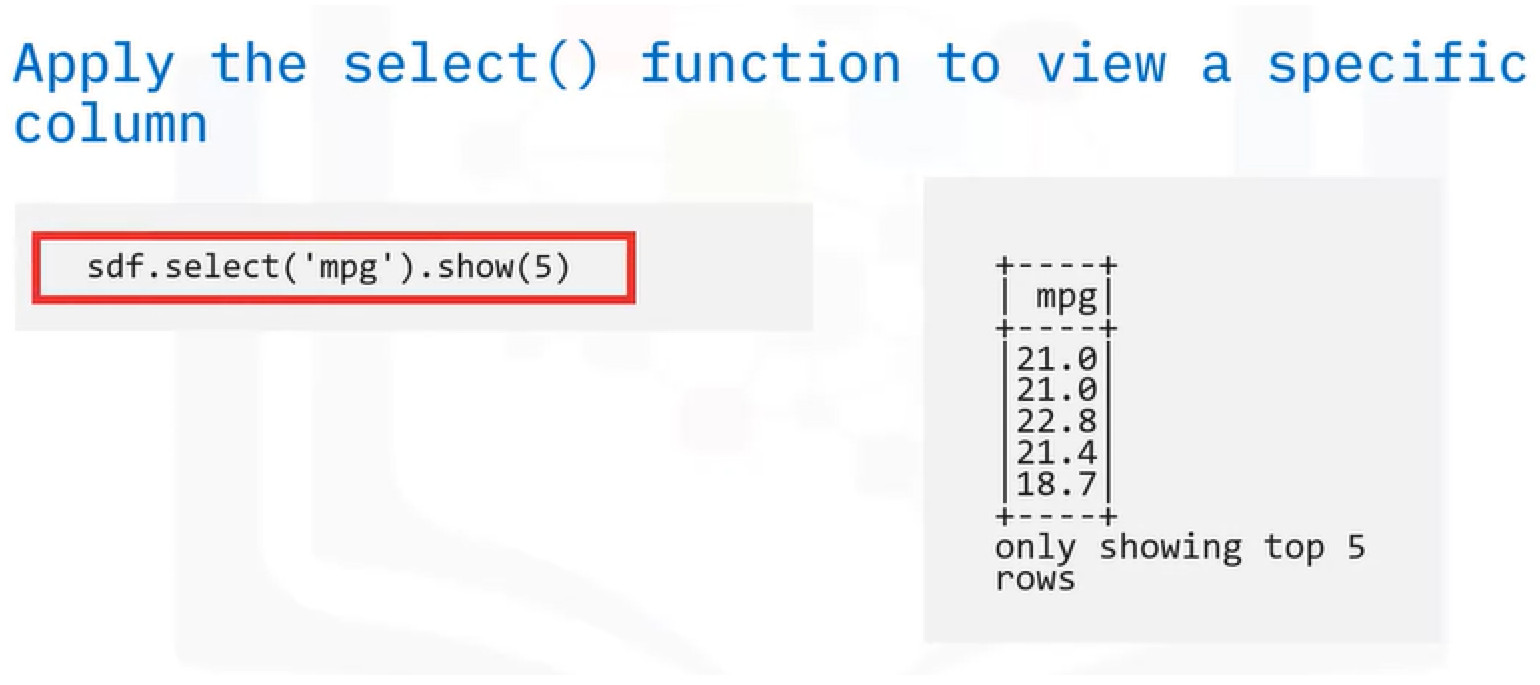
Apply the show() function

```
sdf.show(5)
```

```
+-----------+----+---+-----+---+----+-----+----+--+--+----+----+
| Unnamed: 0| mpg|cyl| disp| hp|drat|   wt|qsec|vs|am|gear|carb|
+-----------+----+---+-----+---+----+-----+----+--+--+----+----+
|  Mazda RX4|21.0|  6|  160|110|3.90|2.620|16.4| 0| 1|   4|   4|
|           |    |   |     |   |    |     |   6|  |  |    |    |
+-----------+----+---+-----+---+----+-----+----+--+--+----+----+
|Mazda RX4 Wag|21.0| 6| 160|110|3.90|2.875|17.0| 0| 1|  4|   4|
|           |    |   |     |   |    |     |   2|  |  |    |    |
+-----------+----+---+-----+---+----+-----+----+--+--+----+----+
| Datsun 710|22.8|  4|108.9| 93|3.85|2.320|18.6| 1| 1|   4|   1|
|           |    |   |     |   |    |     |   1|  |  |    |    |
+-----------+----+---+-----+---+----+-----+----+--+--+----+----+
|Hornet 4 Drive|21.6|6|258.0|110|3.08|3.215|19.4| 1| 0|  3|   1|
|           |    |   |     |   |    |     |   4|  |  |    |    |
+-----------+----+---+-----+---+----+-----+----+--+--+----+----+
|     Hornet|18.7|  8|360.0|175|3.15|3.440|17.0| 0| 0|   3|   2|
| SportAbout|    |   |     |   |    |     |   2|  |  |    |    |
+-----------+----+---+-----+---+----+-----+----+--+--+----+----+
```

Using the select function

Examine data from a specific column in detail.



Apply the select() function to view a specific column

```
sdf.select('mpg').show(5)
```

```
+----+
| mpg|
+----+
|21.0|
|21.0|
|22.8|
|21.4|
|18.7|
+----+
only showing top 5
rows
```

Transform data

Retain only relevant data.

Transformation techniques can include filtering the data, joining with other data sources, or performing columnar operations.

Group or aggregate data

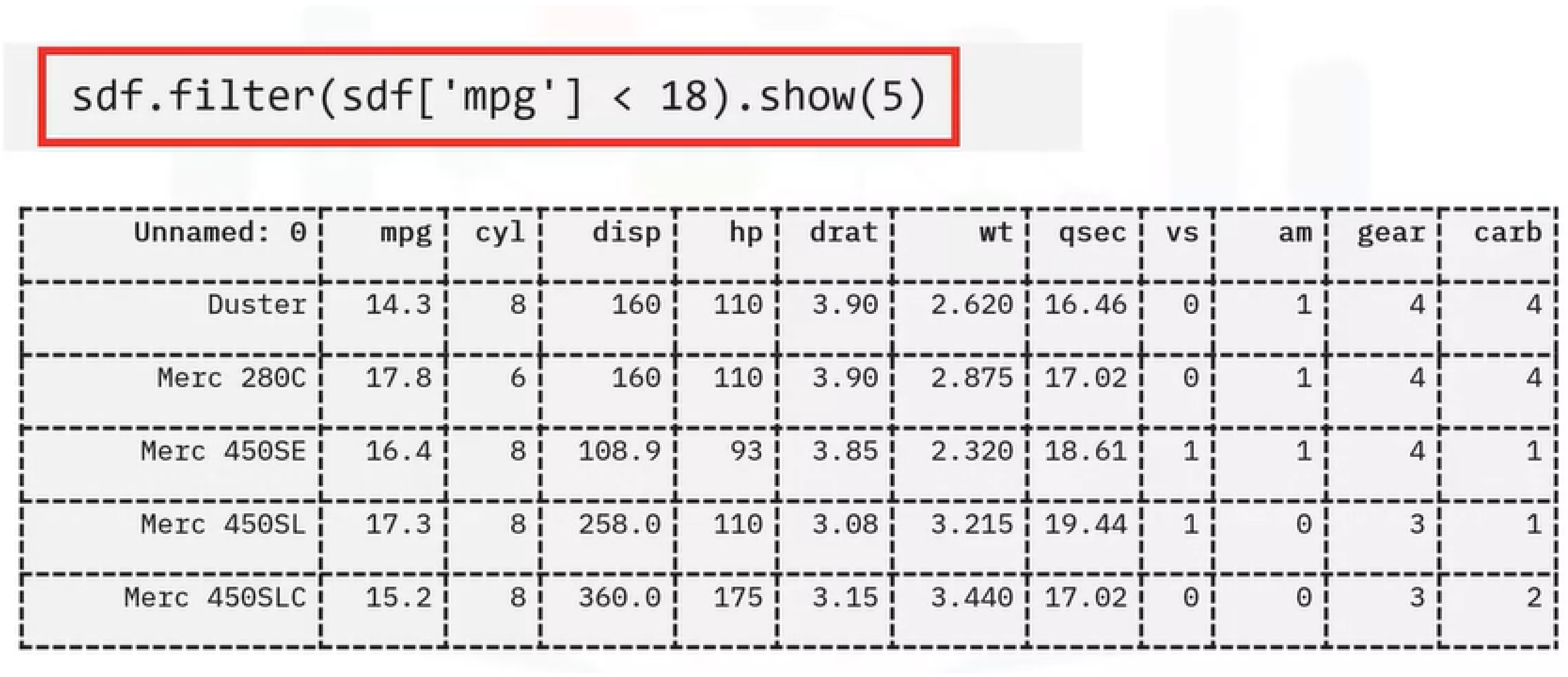
Apply domain-specific data augmentation processes.

💡 ***Columnar operations*** are actions such as multiplying each column by a specific number, converting data from one unit to another, such as converting miles per gallon to kilometers per lite

💡 The effort needed can vary depending on the domain and the data.

For example, scientific datasets involving multiple, differing measurements may require more columnar operations and unit conversions, while financial data can require more aggregation and averaging tasks.

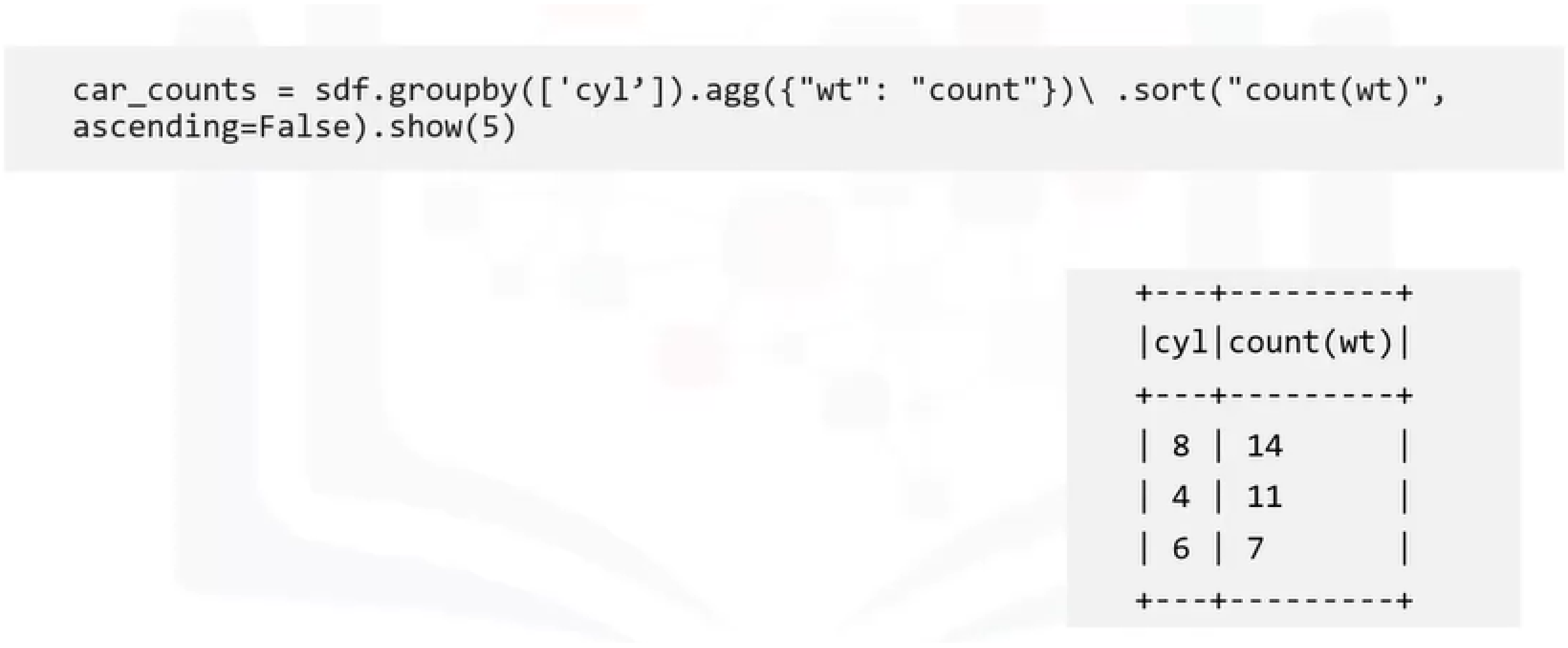
The filter function helps locate specific criteria within the dataset.



```
sdf.filter(sdf['mpg'] < 18).show(5)
```

```
     Unnamed: 0 | mpg | cyl | disp | hp | drat |    wt | qsec | vs | am | gear | carb
         Duster | 14.3|   8 |   160| 110| 3.90 | 2.620 | 16.46|  0 |  1 |    4 |    4
      Merc 280C | 17.8|   6 |   160| 110| 3.90 | 2.875 | 17.02|  0 |  1 |    4 |    4
     Merc 450SE | 16.4|   8 | 108.9|  93| 3.85 | 2.320 | 18.61|  1 |  1 |    4 |    1
     Merc 450SL | 17.3|   8 | 258.0| 110| 3.08 | 3.215 | 19.44|  1 |  0 |    3 |    1
    Merc 450SLC | 15.2|   8 | 360.0| 175| 3.15 | 3.440 | 17.02|  0 |  0 |    3 |    2
```

An example of aggregating data



```
car_counts = sdf.groupby(['cyl’]).agg({"wt": "count"})\ .sort("count(wt)",
ascending=False).show(5)
```

```
+---+---------+
|cyl|count(wt)|
+---+---------+
| 8 | 14      |
| 4 | 11      |
| 6 | 7       |
+---+---------+
```

Loading or Export the data

Final step of the ETL pipeline

- Export the data to disk
- Load it into another database.

Can write the data to the disk as a JSON file or save the data into another database such as a Postgres database.

Can also use an API to export data to another database, such as a Postgres database.

Conclusion

- Basic DataFrame operations are reading, analysis, transformation, loading, and writing.
- You can use a Pandas DataFrame in Python to load a dataset.
- You can apply the print schema, select function, or show function for data analysis.
- For transform tasks, keep only relevant data and apply functions such as filters, joins, column operations, grouping and aggregations, and other functions.

Hands-on Lab: Introduction to Data-Frames

https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsNetwork/labs/DataFrames.ipynb

Real-world usage of SparkSQL

Spark SQL - revisited

- Is a Spark module for structured data processing.
- Is used to run SQL queries on Spark DataFrames
- Has available APIs in Java, Scala, Python, and R.

Create View in Spark SQL

The first step to running SQL queries in Spark SQL is to create a table view.

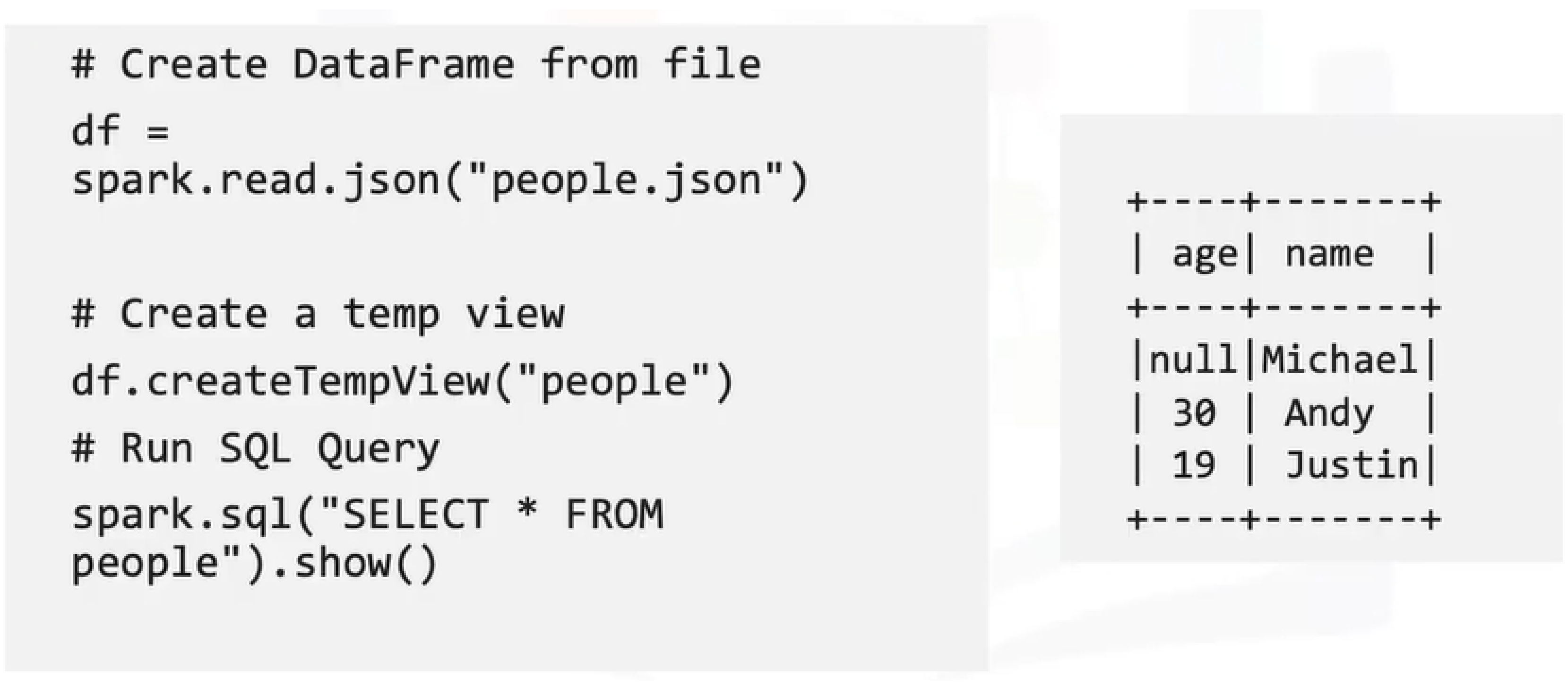
A table view is a temporary table used to run SQL queries.

Spark SQL supports both temporary and global temporary table views.

- A temporary view has local scope.
- A global temporary view is shareable across different Spark sessions.

💡 Local scope means that the view exists only within the current spark session on the current node.
A global temporary view, however, exists within the general Spark application.

Example

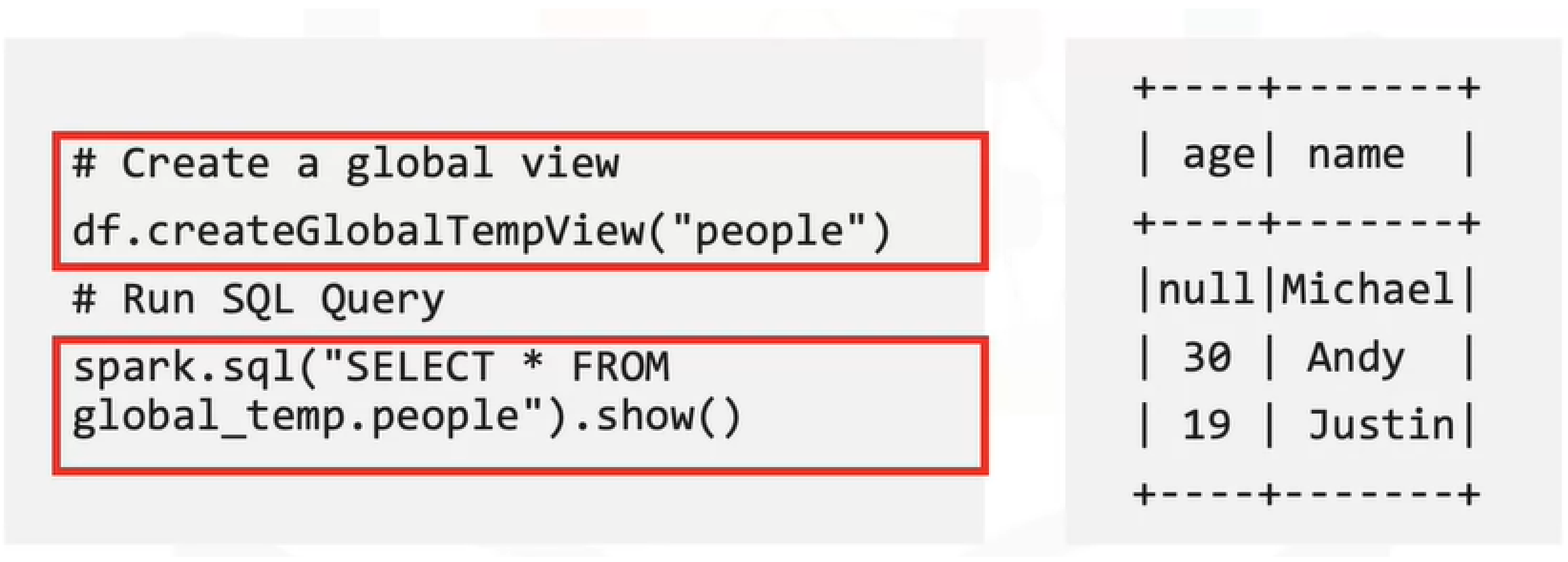


```
# Create DataFrame from file
df =
spark.read.json("people.json")

# Create a temp view
df.createTempView("people")
# Run SQL Query
spark.sql("SELECT * FROM
people").show()
```

```
+----+-------+
| age| name  |
+----+-------+
|null|Michael|
| 30 | Andy  |
| 19 | Justin|
+----+-------+
```

Note the minor syntax change, including the "Global" prefix to the function name and the "global temp" prefix to the view name.



```
# Create a global view
df.createGlobalTempView("people")
# Run SQL Query
spark.sql("SELECT * FROM
global_temp.people").show()
```

```
+----+-------+
| age| name  |
+----+-------+
|null|Michael|
| 30 | Andy  |
| 19 | Justin|
+----+-------+
```

Aggregate data

Used to present grouped statistics.

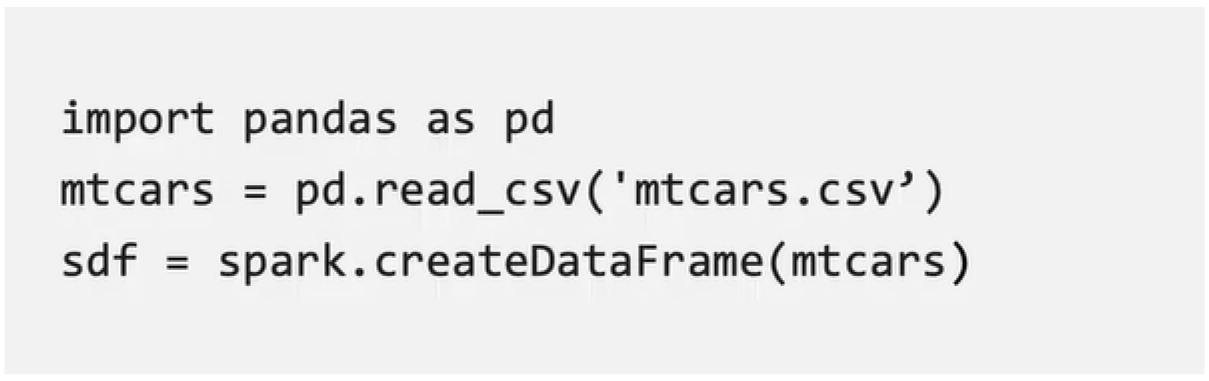
DataFrames come inbuilt with commonly used aggregation functions such as: *count, average, max, and others.*

Can also perform aggregation programmatically using SQL queries and table views.

Example

First, import your data into a DataFrame.

Then, use pandas on Python to read the CSV file and create a DataFrame.

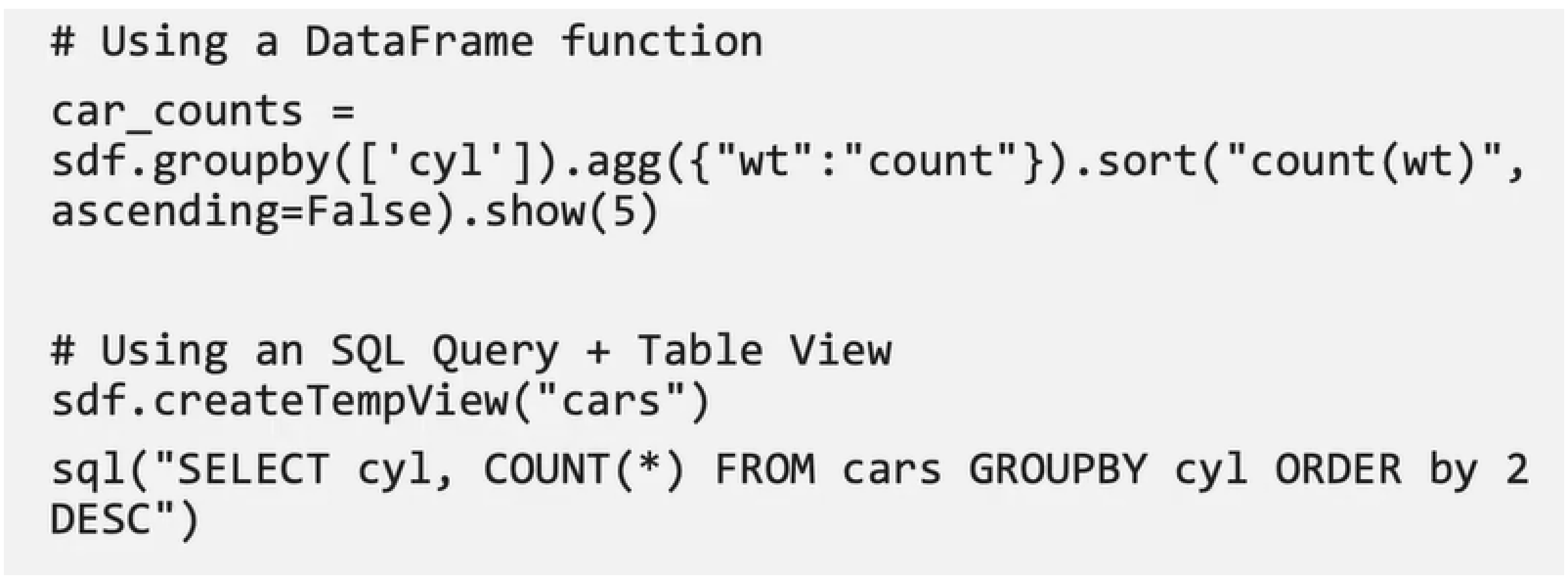


```
import pandas as pd
mtcars = pd.read_csv('mtcars.csv’)
sdf = spark.createDataFrame(mtcars)
```

In this example, you aggregate and group cars by the number of cylinders they have.

Perform this action using two methods:

1. Use the inbuilt functions of the DataFrame.
2. Create a temp view for the DataFrame.



```
# Using a DataFrame function
car_counts =
sdf.groupby(['cyl']).agg({"wt":"count"}).sort("count(wt)",
ascending=False).show(5)


# Using an SQL Query + Table View
sdf.createTempView("cars")
sql("SELECT cyl, COUNT(*) FROM cars GROUPBY cyl ORDER by 2
DESC")
```

Spark SQL data sources

- Parquet Files:

    - is a columnar format supported by many data processing systems.
    - Spark SQL supports reading and writing data from Parquet files,
    - Spark SQL preserves the data schema.
- Spark SQL can load and write to JSON datasets by inferring the schema.
- Spark SQL also supports reading and writing data stored in Hive.

Conclusion

- Spark modules for structured data processing can run SQL queries on Spark DataFrames and are usable in Java, Scala, Python, and R.
- Spark SQL supports both temporary views and global temporary views.
- You can use a DataFrame function or a SQL Query plus Table View for data aggregation.
- Spark SQL supports Parquet files, JSON datasets, and Hive tables.

Labs

Hands-On Lab: Introduction to SparkSQL: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsNetwork/labs/SparkSQL.ipynb

---

Summary & Highlights

- RDDs are Spark's primary data abstraction partitioned across the nodes of the cluster. Transformations leave existing RDDs intact and create new RDDs based on the transformation function. With a variety of available options, apply functions to transformations perform operations. Next, actions return computed values to the driver program. Transformations undergo lazy evaluation, meaning they are only evaluated when the driver function calls an action.
- A dataset is a distributed collection of data that provides the combined benefits of both RDDs and SparkSQL. Consisting of strongly typed JVM objects, datasets make use of DataFrame typesafe capabilities and extend object-oriented API capabilities. Datasets work with both Scala and Java APIs. DataFrames are not typesafe. You can use APIs in Java, Scala, Python. Datasets are Spark's latest data abstraction.
- The primary goal of Spark SQL Optimization is to improve the run-time performance of a SQL query, by reducing the query’s time and memory consumption, saving organizations time and money. Catalyst is the Spark SQL built-in rule-based *query* optimizer. Catalyst performs analysis, logical optimization, physical planning, and code generation. Tungsten is the Spark built-in cost-based optimizer for CPU and memory usage that enables cache-friendly computation of algorithms and data structures.
- Basic DataFrame operations are reading, analysis, transformation, loading, and writing. You can use a Pandas DataFrame in Python to load a dataset and apply the print schema, select function, or show function for data analysis. For transform tasks, keep only relevant data and apply functions such as filters, joins, column operations, grouping and aggregations, and other functions.
- Spark SQL consists of Spark modules for structured data processing that can run SQL queries on Spark DataFrames and are usable in Java, Scala, Python and R. Spark SQL supports both temporary views and global temporary views. Use a DataFrame function or an SQL Query + Table View for data aggregation. Spark SQL supports Parquet files, JSON datasets and Hive tables.

Quiz

Practice Quiz: Introduction to Data-Frames & SparkSQL

Question 1

Transformations in Spark are considered *lazy* because transformation results are not computed right away. Which answer describes why this statement is true?

- Results are computed only by external programs.
- Instead, the results are only computed when evaluated by “functions.”

- **The results are only computed when evaluated by “actions.”**
- The results are only computed by downstream programs.

Question 2

Select the characteristics of datasets.

- **Strongly-typed; use unified Java and Scala APIs; built on top of DataFrames; are the latest data abstraction added to Spark.**
- Strongly-typed; use APIs in Java, Scala, Python and R; built on top of DataFrames; added in earlier Spark versions.
- Strongly-typed; use APIs in Java, Scala, Python and R; built on top of DataFrames; are the latest data abstraction added to Spark.
- Strongly-typed; use APIs in Java, Scala, Python and R; built on top of RDDs; are the latest data abstraction added to Spark.

Question 3

Select the answer that matches the four major phases of Catalyst query optimization.

- Logical Planning; Logistical Optimization; Physical Planning and Code Generation
- Logical Planning; Logistical Optimization; Physical Planning and Physical Optimization
- Analysis; Logistical Optimization; Physical Planning and Evaluation of outcomes
- **Analysis; Logistical Optimization; Physical Planning and Code Generation**

Question 4

What is the primary goal of data transformation?

- To reformat the data for new databases.
- To verify that the data originates from a reputable source.
- **To keep only relevant data.**
- To change the data into a format useful for business users.

Question 5

Using a table view is one way to run SQL queries on data using Spark. Which of the following are true about table views?

- [x] ~~A global temporary view exists within the general Spark application and shareable across different Spark sessions.~~
- [x] ~~A temporary view has local scope, which means that the view exists only within the current Spark session on the current node.~~
- [ ] A temporary view has local scope, which means that the view exists and is sharable across different Spark sessions.
- [ ] A global temporary view exists within the general Spark application and is only available for the current Spark session.

Graded Quiz: Introduction to Data-Frames & SparkSQL

1.Question 1

Select the statements that are true about a Directed Acyclic Graph, known as a DAG.

- [x] ~~A DAG is a data structure with edges and vertices~~
- [x] ~~In Apache Spark, RDDs are represented by the vertices of a DAG while the transformations and actions are represented by directed edges.~~
- [x] ~~If a node goes down, Spark replicates the DAG and restores the node.~~
- [ ] A DAG is a tabular data structure with row and columns
- [x] ~~Every new edge of a DAG is obtained from an older vertex.~~

2.Question 2

Which function would you apply to create a dataset from a sequence?

- **toDS()**
- DSRdd()
- seqDS()
- Create()

3.Question 3

Which of the following features belong to Tungsten?

- [x] ~~**Places intermediate data in CPU registers**~~
- [ ] Prohibits Loop unrolling.
- [ ] Generates virtual function dispatches
- [x] ~~**Manages memory explicitly and does not rely on the JVM object model or garbage collection.**~~

4.Question 4

Select the answer that lists the order in which a typical data engineer would perform operations on Apache Spark while adhering to best practices.

- Analyze, Read, Load, Transform, and Write
- Read, Analyze, Load, Transform, and Write
- **Read, Analyze, Transform, Load and Write**
- Analyze, Read, Transform, Load and Write

5.Question 5

Based on the lesson content, what data sources does Apache Spark SQL support natively?

- [x] ~~**Parquet files**~~
- [x] ~~**JSON files**~~
- [ ] NoSQL databases
- [x] ~~**Hive tables**~~