Spark
SQL

2023

<lapti>

Agenda

- Spark SQL Engine
  - Spark como um SGBD
- Tabelas gerenciadas
- Metastore
- Acessando DataFrames em SQL
- Spark SQL em aplicações

SQL

- Bancos de dados relacionais e SQL estão disponíveis a algumas décadas…
  - Mais do que a idade da maioria de vocês…
  - Gerações de programadores treinados para isso
- Operações em DataFrames são particularmente parecidas…
  - Métodos select(), where(), groupBy(), orderBy() ??
    - Porque não escrever em SQL de uma vez??
- Já no Spark 1.x havia interesse em dar acesso à RDDs via SQL
  - SchemaRDDs
  - Spark Shark

SHARK

Shark

Hive on Spark

Cliff Engle, Antonio Lupher, Reynold Xin, Matei Zaharia, Michael Franklin, Ion Stoica, Scott Shenker

amplab

Agenda

- Intro to Spark
- Apache Hive
- Shark
- Shark's Improvements over Hive
- Demo
- Alpha status

SQL

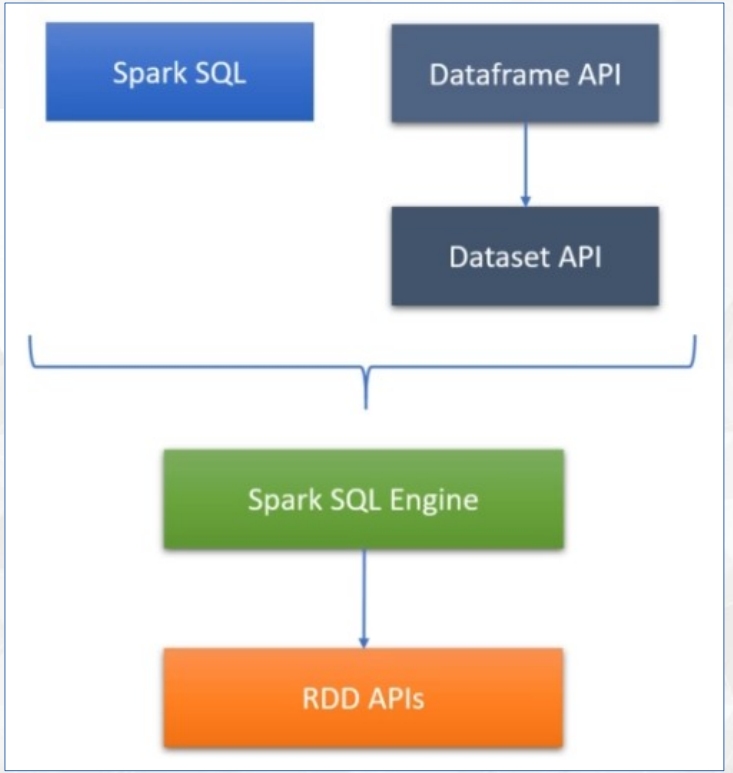
- Então, não seria interessante criar uma camada sobre os DataFrames para poder rodar consultas SQL com esses dados?

Spark SQL

- Módulo para processamento de dados estruturados
  - Através de SQL
- Fornece uma abstração de programação
  - DataFrames
- Também pode ser executado como um **query engine SQL distribuído**
- Alto desempenho
  - Queries Hive rodam 100x mais rápido no mesmo ambiente
- Bem integrado com o restante do ecosistema Spark

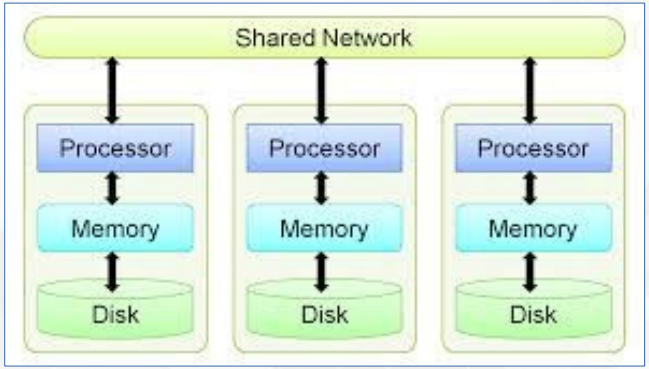
Spark SQL

- Suporte nativo SQL a dados gerenciados pelo Spark
  - RDDs e fontes externas
    - Arquivos, JDBC, conectores, etc
  - Permite usar SQL em dados locais e externos, na mesma aplicação
- Inclui diversos componentes
  - Otimizador baseado em custo
  - Storage colunar
  - Gerador de código
    - Para tornar queries mais rápidas



MPP Databases

- MPP
  - Massively Parallel Processing database
  - Engine de banco de dados otimizado para processamento paralelo
    - Muitas operações executando em muitas unidades de processamento a um dado momento
  - Coordenação do processamento de uma tarefa
    - Múltiplos processadores trabalhando em diferentes partes da tarefa e em diferentes partes dos dados
    - Cada processador tem seu próprio sistema operacional e memória



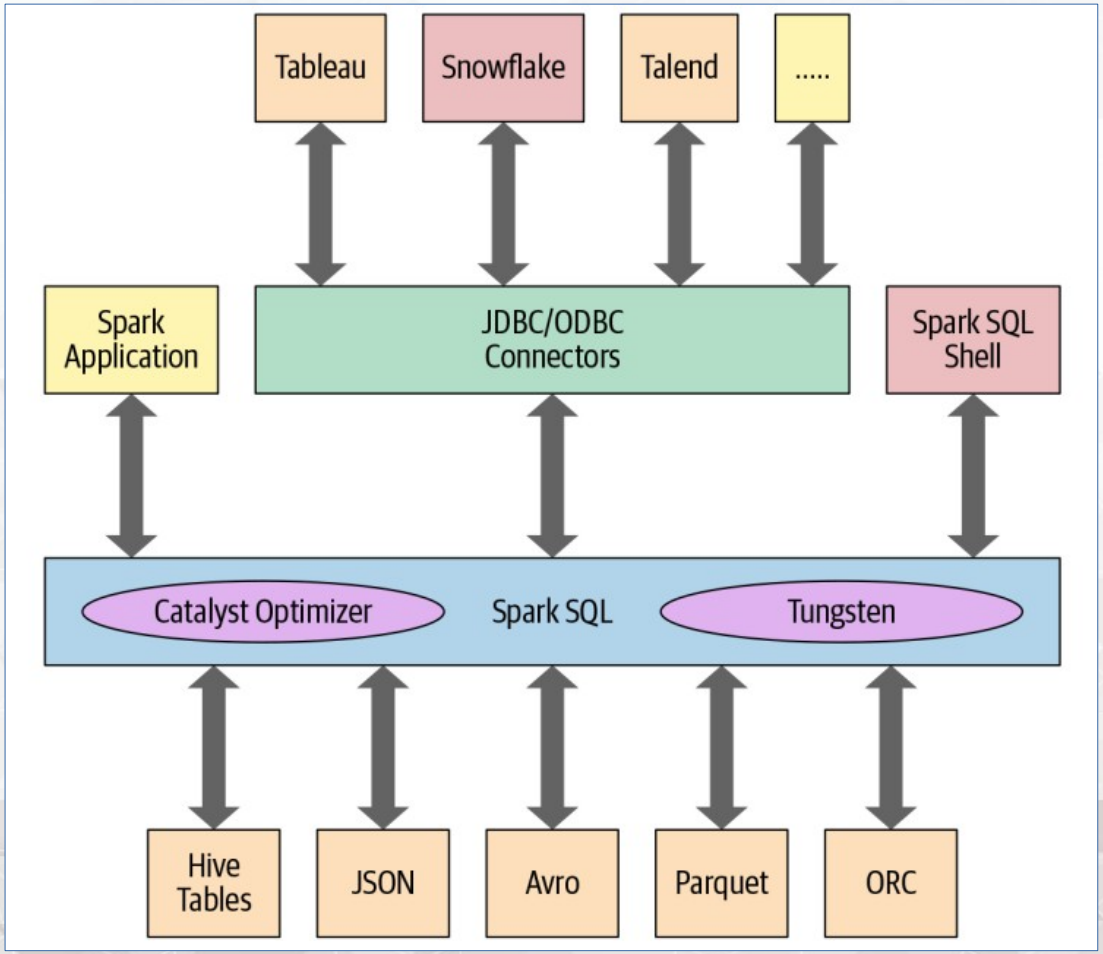
Spark SQL Engine

- DataFrames e Datasets são baseados em RDDs
  - São decompostos em código RDD compacto
    - Geração de código compacto e eficiência de queries
  - Spark SQL Engine realiza essa tarefa
    - Construir queries eficientes e gerar código compacto
- Spark SQL introduzido na versão 1.3
  - Como uma maneira de executar SQL sobre RDDs ou SchemaRDDs
  - Entretanto, evoluiu para um engine generalizado para funções estruturadas de alto nível

Spark SQL Engine

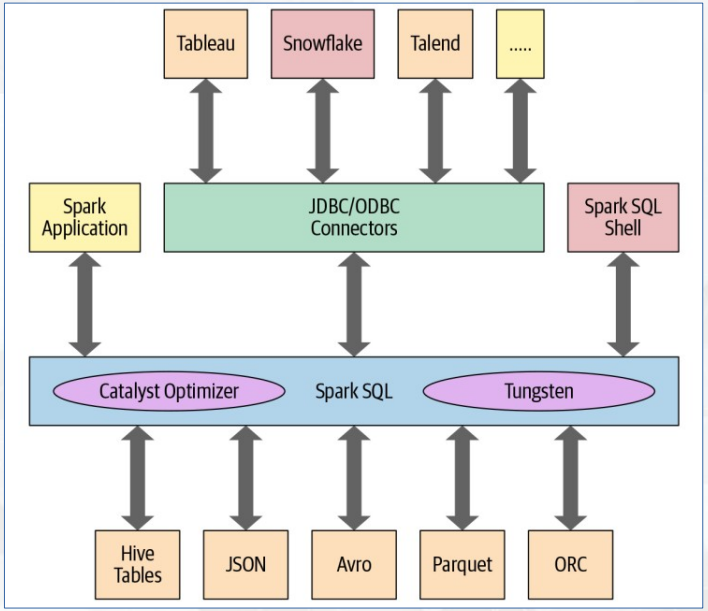
- Várias funcionalidades, além de processar queries em formato SQL
  - Unificar componentes Spark e permitr abstrações de DataFrames/Datasets para as linguagens específicas
    - Java, Scala, Python, R
      - Simplifica o acesso a datasets estruturados
  - Conectar com metastores e tabelas do Apache Hive
  - Ler e gravar dados estruturados com um esquema específico de e para formatos estruturados
    - JSON, CSV, Text, Avro, Parquet, ORC
    - Converte para tabelas temporárias
  - Spark SQL shell para exploração de dados
  - Fornece uma ponte de e para ferramentas externas com drivers padronizados JDBC/ODBC
  - Gera query plans otimizados para a JVM
  - Suporta SQL:2003 ANSI e HiveQL

Spark SQL Engine



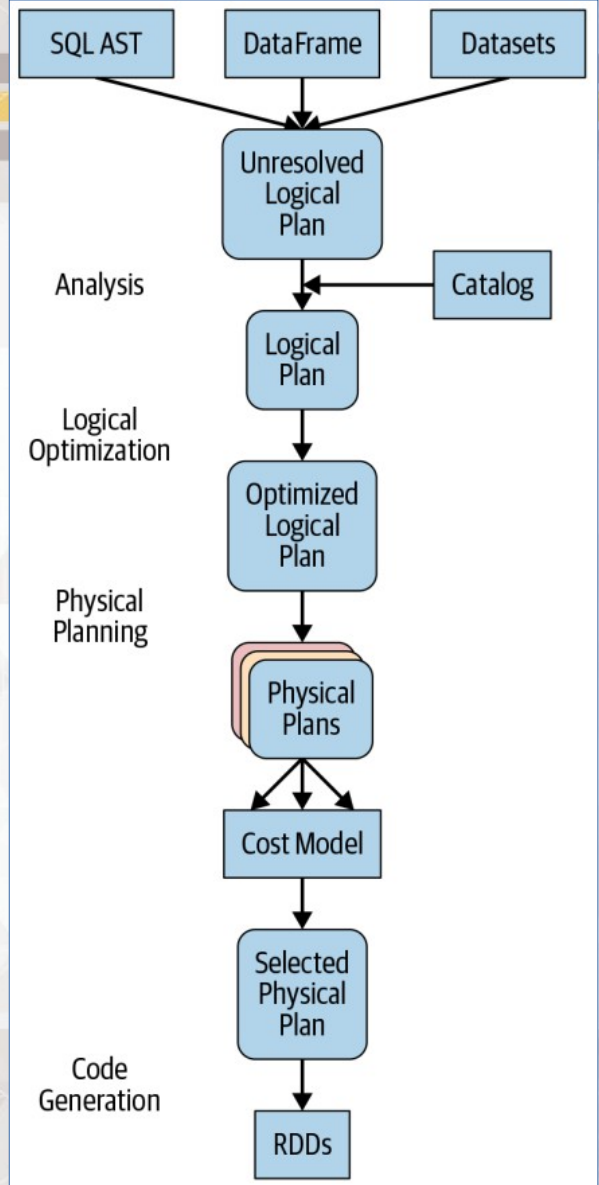
Spark SQL Engine

- Tungsten
  - Engine para dados in-memory
  - Encoders, off-heap memory, vectorized column-based memory layout, acesso paralelo a dados, etc
- Catalyst Optimizer
  - Recebe uma query e converte em um plano de execução
    - Análise
    - Otimização lógica
    - Planejamento físico
    - Geração de código



Spark SQL Engine

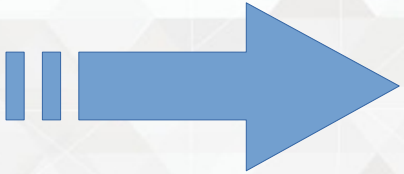
- Catalyst
  - Quatro fases transformacionais
    - Análise
    - Otimização lógica
    - Planejamento físico
    - Geração de código
  - Etapas podem ser visualizadas com o método explain
    - DataFrame.explain(true)



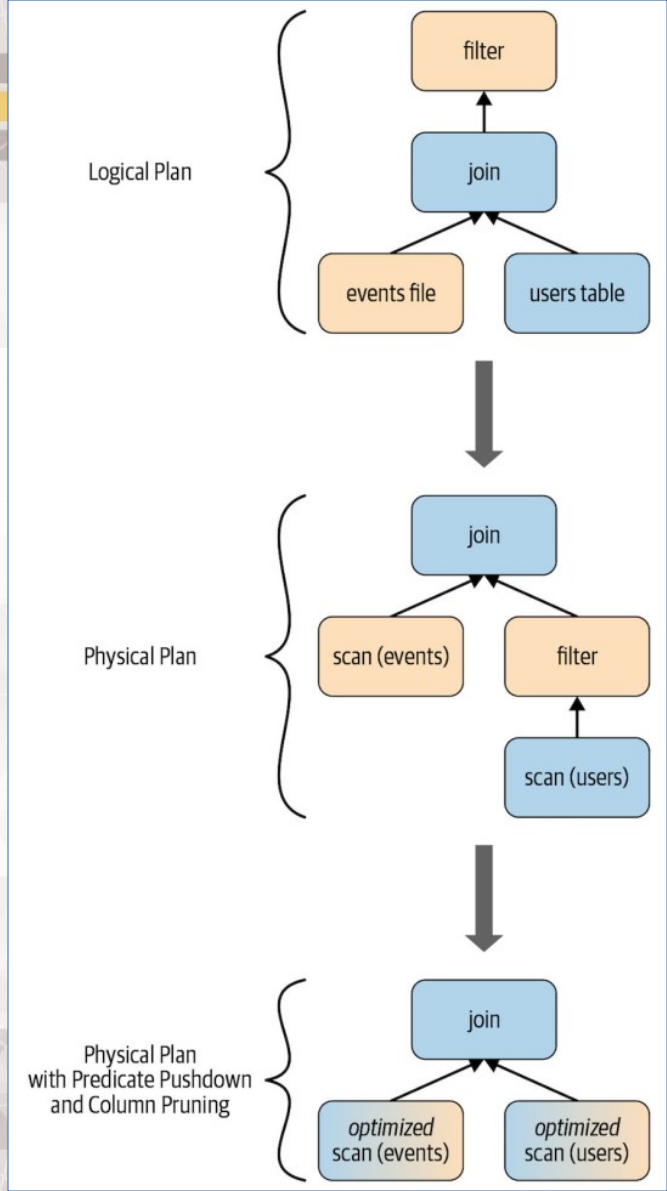
Spark SQL Engine

```
users = spark.read.csv('users.csv', header = True, inferSchema = True)
events = spark.read.csv('events.csv', header = True, inferSchema = True)

userEvents = users.join(events, users('id') == events('uid'))
        .filter(events('date') > '2021-01-01')
```



```
select id, uid, event_name
    from users join events on (users.id = events.uid)
    where events.date > '2021-01-01'
```

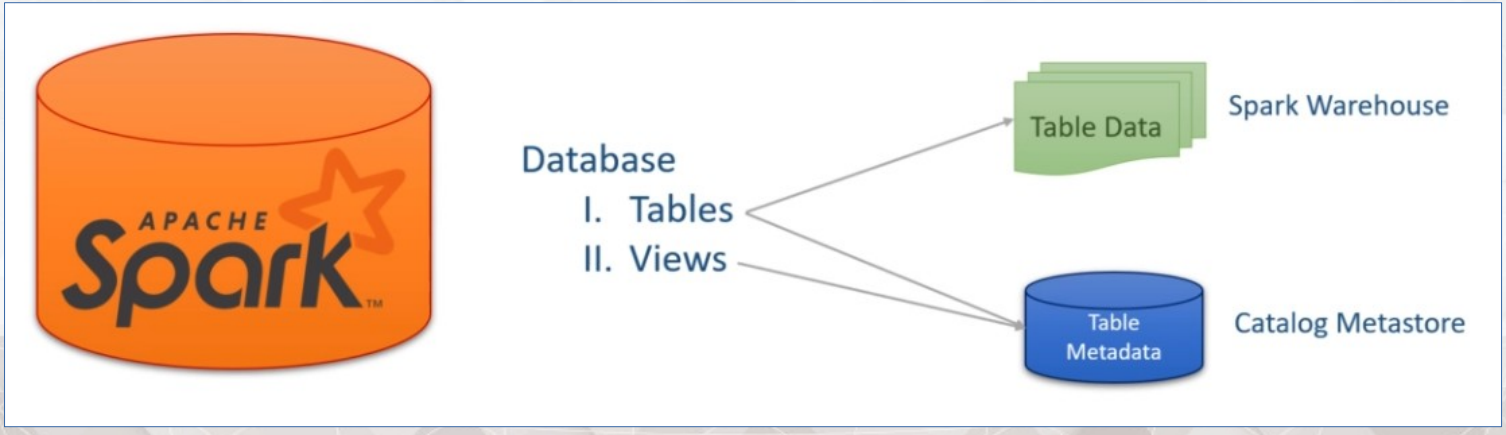


Tabelas e Views

- Spark SQL possui modos de funcionamento
  - SQL engine distribuído
    - Gerenciando um datastore próprio
    - Tabelas locais ou externas
    - Dados podem ser importados para tabelas Spark
  - Acessando dados externos
    - Usando dados disponíveis em arquivos externos
    - SQL engine para arquivos
    - Dados expostos como “temporary views”

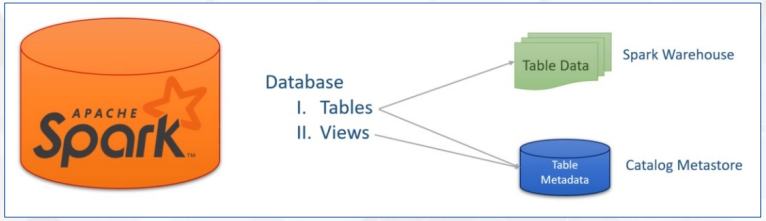
Tabelas e Views

- Tabelas managed X unmanaged
  - Tabelas mantém dados e metadados relevantes
- Apache Spark usa Apache Hive metastore
  - Em */user/hive/warehouse*
    - Localização configurada em spark.sql.warehouse.dir



Tabelas e Views

- Managed table
  - Spark gerencia dados e metadados
    - Dados em formato interno do Spark (parquet)
  - Localização em Spark Warehouse
- Unmanaged table
  - Spark gerencia somente os metadados
    - Arquivos são externos e gerenciados pelo usuário
      - Incluindo data sources externos, como Cassandra



Tabelas e Views

- Dados de DataFrames podem ser acessados via SQL com views
  - Temporary views
  - Global temporary views
- Tabelas gerenciadas pelo Spark tem melhor desempenho
  - Principalmente em processamento de maior volume de dados
    - Paralelização e uso dos recursos de um cluster

Spark como SQL Engine

- Spark se comporta como um banco de dados MPP
  - Usando managed tables
  - E Hive Metastore
- Comandos convencionais para criação e gerenciamento
  - Gerenciamento de um BD Spark
  - De estruturas e de dados
- Acesso via APIs ou drivers

Spark como SQL Engine

- Criando bancos de dados e tabelas

```
spark.sql("CREATE DATABASE employees_db")
spark.sql("USE employees_db")
```

```
spark.sql("CREATE DATABASE IF NOT EXISTS emp1");
spark.catalog().setCurrentDatabase("emp1");
```

Spark como SQL Engine

- Criando e populando tabelas

```
# managed table
spark.sql("CREATE TABLE employees (emp_no INT, first_name STRING, birth_date DATE)")



#unmanaged table
spark.sql("""CREATE TABLE delay_flights(date STRING, delay INT,
    distance INT, origin STRING, destination STRING)
    USING csv OPTIONS (PATH '/var/datasets/departuredelays.csv')""")
```

Spark como SQL Engine

- Criando e populando tabelas

```
# criando a partir de um DataFrame
arquivo = "/var/datasets/departuredelays.csv"

schema="date STRING, delay INT, distance INT, origin STRING, destination STRING"

df = spark.read.csv(arquivo, schema=schema)
df.write.saveAsTable("delay_flights")
```

Spark como SQL Engine

- Criando e populando tabelas

```
String schemaEmployees = "emp_no INT, birth_date DATE, first_name STRING, "
        + "last_name STRING, gender STRING, hire_date DATE";

Dataset<Row> df = spark.read().option("header", true).schema(schemaEmployees)
        .csv("/var/datasets/employees/employees.csv");

df.write().mode("overwrite").saveAsTable("employee");
```

Spark SQL em Aplicações

- SparkSession
  - Spark 2.0
  - Unified entry point
  - Método sql para execução de queries
    - `spark.sql("select * from tabela1")`
    - Queries resultam em um DataFrame
      - Que podem ter os mesmos métodos convencionais aplicados
  - Desempenho pode ser diferente entre tabelas e views
    - Também dependendo do tipo de arquivo para armazenamento

Spark SQL em Aplicações

- Carregando dados para um DataFrame e executando queries em SQL

```
String schemaEmployees = "emp_no INT, birth_date DATE, first_name STRING, "
        + "last_name STRING, gender STRING, hire_date DATE";

Dataset<Row> df = spark.read().option("header", true).schema(schemaEmployees)
        .csv("/home/leandro/Documentos/datasets/employees/employees.csv");


df.createOrReplaceTempView("employee");

spark.sql("select * from employee").show();
```

Spark SQL em Aplicações

- Views tem sintaxe de criação semelhante a tabelas
  - Mas podem ser criadas a partir de tabelas ou de outras fontes de dados
    - Como DataFrames já existentes
- Usar um schema pré-definido potencialmente melhora o desempenho das queries
  - Aumenta a chance de otimização pelo query engine processor
    - Catalyst
- Views tem desempenho pior que tabelas
  - Mas são mais versáteis para programação de aplicações e exploração
    - Tabelas armazenadas em um datastore Spark geralmente são baseadas em dados mais estáveis ou permanentes

Spark SQL em Aplicações

- Views podem ser locais ou globais
  - Temporary
  - Global Temporary
- Locais
  - Criadas e visíveis a partir de uma única SparkSession
- Globais
  - Visíveis em todas as SparkSessions de uma aplicação

Spark SQL em Aplicações

- Cache de dados
  - Spark já procura fazer cache de dados mais utilizados
    - Bem como resultados intermediários, tabelas mais acessadas, etc
  - Mas aplicação pode solicitar cache para tabelas e views
    - Pode também solicitar cache para DataFrames, por falar nisso…
  - Cache pode ser LAZY
    - Cacheado somente na primeira utilização do objeto
  - Ex.:
    - `spark.sql("cache table employee");`

Fontes de dados externas

- Spark pode interagir com diversas fontes externas de dados
  - Thrift servers
  - JDBC
    - E connectors resultantes, como ODBC
    - JDBC driver deve estar no classpath
      - Ou apontado por funções do sistema em Python e R
  - Apache Cassandra
  - Snowflake datawarehouse
  - MongoDB

Fontes de dados externas

- Options em DataFrame
  - driver
  - url
  - user, password
  - dbtable ou query
- Driver deve estar disponível no classpath

```
Dataset<Row> datasetMySQL = sparkMySQL.read().format("jdbc")
        .option("url", "jdbc:mysql://localhost/employees?user=root&password=1234")
        .option("dbtable","employees")
        .option("driver","com.mysql.jdbc.Driver")
        .load();
```

Fontes de dados externas

- Particionamento
  - Transferência de grande volume de dados pode saturar recursos da fonte
    - Ou do destino, aliás
  - Conexões paralelas serão criadas para suportar o particionamento
- Conexão externa permite parâmetros para particionamento
  - numPartitions
  - partitionColumn
  - lowerBound, upperBound

Então...

- Spark SQL é uma maneira simples e fácil para realizar consultas SQL em DataFrames Spark
  - Geralmente mais simples usar a sintaxe bem conhecida do SQL para realizar transformações em dados
- Mas vai muito além disso
  - Conexão com diversas fontes externas
  - E principalmente, pode ser um SQL engine distribuído em um cluster Spark
    - MPP
  - Combinações e operações entre diferentes fontes de dados
    - DataFrames, arquivos, JDBC, NoSQL, Hive, Hadoop, etc

Obrigado

leandro@utfpr.edu.br

<lapti>