LOGO

Spark SQL

董炫辰

简介

Spark SQL is a Spark module for structured data processing and can also act as distributed SQL query engine.

It provides a programming abstraction called DataFrames .

相关组件

shark：

Shark已经不会再投资资源去更新了

Spark SQL利用了**Shark**中最好的部分**(**如列式存储**)**

hive：

支持用**hql**来写查询语句

兼容**metastore**（这表明在**shark**和**hive**中建的表，在**spark sql**中都可以直接使用）

列式存储

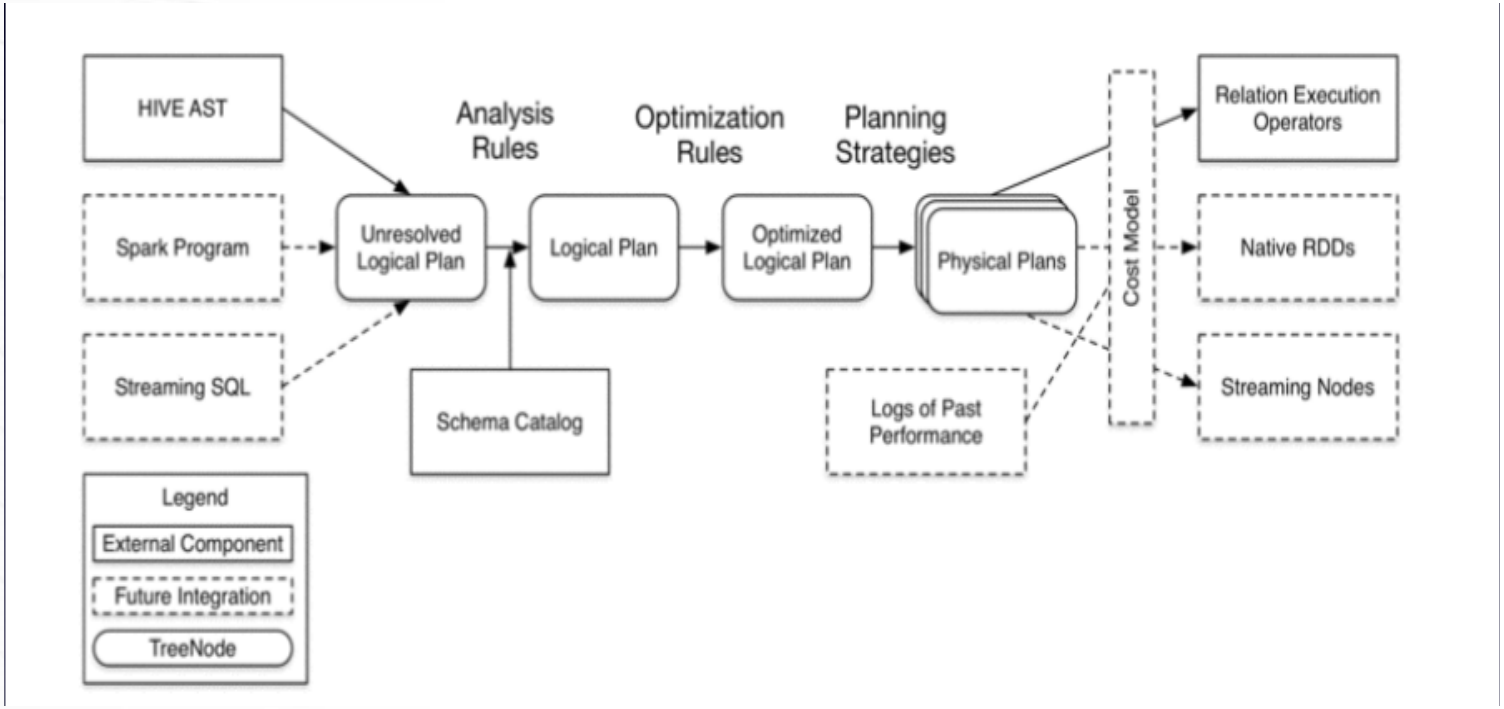
行式存储，是把一行数据作为一个整体来存储。
列式存储，每个列单独作为一个**cell**来存储。

优势

1.每一列数据都是同类型的，压缩算法很高效。

2.仅需访问感兴趣的相关列的子集，而避免了扫描整个表，这样可以减少磁盘的**I/O**、提升缓存利用率。

架构


HIVE AST
Spark Program
Streaming SQL
Analysis Rules
Unresolved Logical Plan
Schema Catalog
Logical Plan
Optimization Rules
Optimized Logical Plan
Planning Strategies
Physical Plans
Logs of Past Performance
Cost Model
Relation Execution Operators
Native RDDs
Streaming Nodes
Legend
External Component
Future Integration
TreeNode

DataFrame

- **A DataFrame is a distributed collection of data organized into named columns**

- 使用前的准备：

```
val sc: SparkContext
val sqlContext = neworg.apache.spark.sql.SQLContext(sc)
import sqlContext.implicits._
```

LOGO

DataFrame

--创建dataframe

```
val df = sqlContext.read.json(
        "examples/src/main/resources/people.json")
```

- sqlContext.sql("SELECT field1 AS f1, field2 as f2 from table1")
- sqlContext.jsonRDD(json)
- sqlContext.createDataFrame(l)//这个可以是RDD或者序列

LOGO

DataFrame

--dataframe的相关操作

```
df.show()
// age  name
// null Michael
// 30   Andy
// 19   Justin
```

people.json

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

```
df.printSchema()
// root
// |-- age: long (nullable = true)
// |-- name: string (nullable = true)
```

```
df.select("name").show()
// name
// Michael
// Andy
// Justin
```

LOGO

DataFrame

--dataframe的相关操作

```
df.select(df['name'], df['age'] + 1).show()
// name    (age + 1)
// Michael null
// Andy    31
// Justin  20

df.filter(df['age'] > 21).show()
// age name
// 30  Andy

df2.filter("tableName = 'table1'").first()

df.groupBy("age").count().show()
// age  count
// null 1
// 19   1
// 30   1
```

people.json

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

SQL

- 把**dataframe**注册成表

```
df.registerTempTable("people")
```

- 把**RDD**注册成表

```
case class Person(name: String, age: Int)
val people=sc.textFile("examples/src/main/resources/people.txt").
map(_.split(",")).map(p => Person(p(0), p(1).trim.toInt)).toDF()
people.registerTempTable("people")
```

```
sqlContext.jsonRDD(lineRdd).registerTempTable(table_name)
```

SQL

- 使用**sql**语句从表中查询数据

```
sqlContext.sql(" SELECT ss.ApplicationID FROM SparkListenerJobStart
ss")

sqlContext.sql("
  SELECT
     ss.ApplicationID, ss.JobID, ss.SubmissionTime as submit,
     floor((se.CompletionTime - ss.SubmissionTime)/1000) as duration,
  FROM
     SparkListenerJobStart ss, SparkListenerJobEnd se
  Where
     ss.ApplicationID = se.ApplicationID And ss.JobID = se.JobID
  Order by submit, duration
  ")
```

SQL

- 在**sql**语句中，**spark sql**还不支持**in/not in**后加子查询。
  但是可以用**LEFT OUTER JOIN**，来实现

```
sqlContext.sql("
SELECT ts.TaskInfo.TaskID as taskId,
        ts.TaskInfo.ExecutorID as executorId,
        ts.TaskInfo.Host as host, ts.TaskInfo.LaunchTime as
        startTime, te.TaskInfo.FinishTime as finishTime
FROM  SparkListenerTaskStart ts
LEFT OUTER JOIN SparkListenerTaskEnd te
ON ts.TaskInfo.TaskID = te.TaskInfo.TaskID AND
ts.TaskInfo.ExecutorID =
te.TaskInfo.ExecutorID " )
```

LeftSemiJoin类似于**sql server** 中的**exists/in** 查询子句

JSON相关

- **JSON(JavaScript Object Notation)**

  易于人阅读和编写，同时也易于

  机器解析和生成(一般用于提升网络传输速率)。

JSON相关

json中有两种表示数据的结构（对象和数组）

1、对象：对象在**js**中表示为**{}**括起来的内容，数据结构为 **{key：value,key：value,...}**

取值方法为 对象**.key** 获取属性值

2、数组：数组在**js**中是中括号**“[]”**括起来的内容，数据结构为 **["java","javascript","vb",...]**，取值方式和所有语言中一样，使用索引获取

```
var people ={
    "people":[
        {"firstName":"Brett","lastName":"McLaughlin","email":"aaaa"},
        {"firstName":"Jason","lastName":"Hunter","email":"bbbb"},
        {"firstName":"Elliotte","lastName":"Harold","email":"cccc"}
    ]
}
people.people[1].lastName // Hunter
```

JSON相关

```
val people = sqlContext.jsonFile("hdfs://server1:9090/xxx.json")
//读进来之后，就是dataframe

people.printSchema
//打印schema信息，即表中字段的信息

people.regesterAsTable("people")

sql("select * from people where name > 10").collect
```

支持嵌套查询 **where a.b.c='xxx'**

JSON和python的互操作

json.dumps()
把一个**Python**对象编码转换成**Json**字符串

json.loads()
把**Json**格式字符串解码转换成**Python**对象

```
json_format = json.loads(item)
json_format['ApplicationID'] = appId
key = json_format['Event']
value = json.dumps(json_format).replace(' ', '')
```

造一个遍历顺序和插入顺序一样的**dict**。

```
dict = collections.OrderedDict()
```

Python	JSON
dict	object
list, tuple	array
str, unicode	string
int, long, float	number
True	true
False	false
None	null

parquet相关

列式存储的组件。

读取的时候，比如需要**name**，就只读**name**这一列，**IO**比较小，压缩什么的也消耗少，**GC**开销小

支持读取**Parquet**中的数据

支持写到**Parquet**中时保存元数据的**schema**信息

列式存储避免读出不需要的数据

parquet相关

使用**parquet**的步骤：

```
xxx.saveAsParquetFile("xxx.parquet")
//存,注意这个接口是dataframe的

val parquetFile = sqlContext.parquetFile("xxx.parquet")
//读

parquetFile.registerAsTable("parquetFile")
 //跟之前完全一样
```

parquet相关

LOGO

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext._ //先引入sqlContext

val data = sc.textFile("绝对路径")
case class Person(name: String, age: Int)
val people = data.map(_.split(',')),map(p => Person(p(0), p(1).toInt))
people.registerAsTable("people")

sqlContext.sql("xxxxx").saveAsParqueetFile("xxx.parquet")
//以列式存储的方式存起来
val parquertFile = sqlContext.parquetFile("xxx.parquet")
parquertFile.registerAsTable("parquertFile")

rs = sql("select name from parquertFile where age > 1and age < 50")
rs.map(n => "name" + n(0)).collect.foreach(println_)
```

LOGO

cached tables

sqlContext.cacheTable("tableName")

列式存储**(**压缩**+**减少内存使**?+**减少**GC)**

uncacheTable("tableName")取消缓存

调用原生**rdd**的**cache**接口无法享受列式存储的优势

相关类

org.apache.spark.sql.Row

```
val row = Row(1, true, "a string", null)
// row: Row = [1,true,a string,null]
val firstValue = row(0)
// firstValue: Any = 1
val fourthValue = row(3)
// fourthValue: Any = null

val firstValue = row.getInt(0)
// firstValue: Int = 1
val isNull = row.isNullAt(3)
// isNull: Boolean = true
```

相关类

org.apache.spark.sql.Column

```
df("colA")

df.select( df("colA")).show()
```

org.apache.spark.sql.GroupedData

```
df.groupBy(df.age).count().collect()
[Row(age=2, count=1), Row(age=5, count=1)]

df.groupBy().max('age').collect()
[Row(max(age)=5)]

df.groupBy().mean('age').collect()
[Row(avg(age)=3.5)]

df.groupBy().sum('age').collect()
[Row(sum(age)=7)]
```

最近做的事情

1.使用**python plotmatlab**图形库，画图，基本就是柱状图，折线图和点图

2.各种**log**的转换，包括**debug log**， **event log**， **master event log**，**pem log**

3.使用**spark sql**， 利用注册的表，查询集群中运行任务的情况，比如整个集群中各个时间段内运行的**task**总数，单个机器各个时间段内运行**task**的总数，**ego**那边起**container**到**spark**那边拿到这个**container**的延迟，可以查到各种各样的事情

下面请看两个例子：
1.把**event log**中的数据读进来，注册成表
2.利用表中的数据，使用**spark sql**，查询出整个集群每个时间段内的**task**总数
（同样的需求，查询所有机器的每个时间段内的**task**总数，然后大家可以想想）

感谢您的关注

谢谢观赏

WPS Office

Make Presentation much more fun