## Udf和缺失值处理

import org.apache.spark.sql.functions.{udf => normalUdf}

import scala.reflect.runtime.universe.TypeTag

/\*\*

\* editor: xuhao

\* date: 2018.03.20 09:00:00

\*/

/\*\*

\* 用于处理sparkSQL中创建udf函数时null的输出问题，使udf函数可以实现null => null的映射

\* ----

\* 描述：

\* sparkSQL中的udf有以下弊端：

\* 输入可能带有null值，但输出不行，数据会发生信息损失。

\* eg,

\* {{{

\* Long(with null) => Double中不能有null => null的映射：

\* udf((s: Long) => s match {

\* case null => null

\* case e => e / width

\* } // not compile

\* }}}

\* ----

\* 示例：

\* NullableFunctions的udf暗含了的null => null的映射，数据不会损失信息

\* eg,

\* {{{

\* NullableFunctions.udf((s: Long) => s / width)

\* }}}

\* ----

\* The main source idea by Martin Senne's answer on Stack OverFlow.

\*/

object NullableFunctions {

def udf[RT: TypeTag, A1: TypeTag](f: Function1[A1, RT])

: UserDefinedFunction = normalUdf[Option[RT],A1](

(i: A1) => i match {

case null => None

case s => Some(f(s))

})

def udf[RT: TypeTag, A1: TypeTag, A2: TypeTag](f: Function2[A1, A2, RT])

: UserDefinedFunction = normalUdf[Option[RT], A1, A2](

(i1: A1, i2: A2) => (i1, i2) match {

case (null, \_) => None

case (\_, null) => None

case (s1, s2) => Some(f(s1, s2))

})

}

## 开窗函数

一个例子

import org.apache.spark.sql.Row

import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}

val lst = Array.range(10, 20, 2).flatMap(i => Array("A", "B", "C", "D").map(x => (x, i)))

val rdd = sc.parallelize(lst).map(Row.fromTuple)

val bindDF = hqlc.createDataFrame(rdd, StructType(Array(StructField("category", StringType), StructField("count", IntegerType))))

bindDF.registerTempTable("tableName")

val sqlExpr = "SELECT `category`, `count` from (SELECT `category`, `count`, rank() OVER (PARTITION BY category ORDER BY count DESC) as rank FROM `tableName`) tmp WHERE rank <= 2"

val resultDF = hqlc.sql(sqlExpr)

resultDF.show()

## Schema的apply方法

Schema中有apply方法注意用，要比fieldIndex在索引省劲

## groupBy之后重命名的问题

1）使用sql语句最方便

2）如果不使用sql语句，下面的方式可以

DF.agg(*sum*(*col*("count")).alias("count"))

## Spark对Where子查询的支持

1）running this query in Spark shell but it gives me error

sqlContext.sql(

"select sal from samplecsv where sal < (select MAX(sal) from samplecsv)"

).collect().foreach(println)

error:

java.lang.RuntimeException: [1.47] failure: ``)'' expected but identifier MAX found

select sal from samplecsv where sal < (select MAX(sal) from samplecsv) ^ at scala.sys.package$.error(package.scala:27) Can anybody explan me,thanks

2）Spark SQL should support both correlated and uncorrelated subqueries. See SubquerySuite for details. Some examples include:

select \* from l where exists (select \* from r where l.a = r.c)

select \* from l where not exists (select \* from r where l.a = r.c)

select \* from l where l.a in (select c from r)

select \* from l where a not in (select c from r)

3)Unfortunately as for now (Spark 2.0) it is impossible to express the same logic using DataFrame DSL.

2.0和2.0+支持一下子查询

select \* from l where exists (select \* from r where l.a = r.c)

select \* from l where not exists (select \* from r where l.a = r.c)

select \* from l where l.a in (select c from r)

select \* from l where a not in (select c from r

2.0-只支持有继承关系的子查询 Hive <= 0.12

SELECT col FROM (SELECT \* FROM t1 WHERE bar) t2

It simply doesn't support subqueries in the WHERE clause.Generally speaking arbitrary subqueries (in particular correlated subqueries) couldn't be expressed using Spark without promoting to Cartesian join.

具体sparkSQL支持的子查询语句可以查看apache/spark项目的

[spark](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48)/[sql](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql)/[core](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql/core)/[src](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql/core/src)/[test](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql/core/src/test)/[scala](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql/core/src/test/scala)/[org](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql/core/src/test/scala/org)/[apache](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql/core/src/test/scala/org/apache)/[spark](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql/core/src/test/scala/org/apache/spark)/[sql](https://github.com/apache/spark/tree/df89f1d43d4eaa1dd8a439a8e48bca16b67d5b48/sql/core/src/test/scala/org/apache/spark/sql)/SubquerySuite.scala文件

4）通过join来实现子查询功能

Since subquery performance is usually a significant issue in a typical relational system and every subquery can be expressed using JOIN there is no loss-of-function here.

示例：

/\*\*

\* 创造一个数据

\* [id, class, name, score]

\*/

val lst = List(

Array(1, 1, "张三", 92),

Array(2, 1, "李四", 70),

Array(3, 1, "王二", 30),

Array(4, 2, "赵武", 100),

Array(5, 2, "周六", 85),

Array(6, 2, "孙七", 40),

Array(7, 3, "吴八", 96),

Array(8, 3, "郑九", 85),

Array(9, 3, "冯十", 65))

val rowRdd = sc.parallelize(lst).map(Row.fromSeq(\_))

val df = sqlc.createDataFrame(rowRdd,

StructType(Array(StructField("id", IntegerType),

StructField("class", IntegerType),

StructField("name", StringType),

StructField("score", IntegerType)

)))

df.show()

df.registerTempTable("testscore")

val subSql\_byWhere = "SELECT `testscore`.\* FROM `testscore` WHERE score = (SELECT MAX(score) from `testscore`)"

val SubSql\_byJoin = "SELECT `testscore`.\* FROM `testscore` JOIN (select max(`score`) as score FROM `testscore`) tmp ON testscore.score = tmp.score"

// val result = sqlc.sql(SubSql\_byJoin) // not compile

val result = sqlc.sql(SubSql\_byJoin)

result.show()

5）为什么不支持子查询

我估计可能和lazy模式有关，子查询的模式可能和lazy有关，WHERE子查询如果实现应该需要获取数据的，即出发action的，而spark.sql只是transformation操作，由于某种机制（估计，这个我也不清楚两者暂时不能混在一起）。