第5章Spark SQL 实训

CH05实训1：统计分析某公司每年的产品销售量及销售额

训练要点

1. RDD转化为DataFrame的方法
2. DataFrame的查询操作

需求说明

以某公司的交易数据，该数据包含3个.txt文档数据，分别为日期数据tbData.txt，订单表头数据tbStock.txt和订单明细tbStockDetail.txt.

将tbData.txt,tbStock.txt,tbStockDetail.txt上传到 HDFS，通过sparkContext读取该数据得到一个RDD，将该RDD数据转换成DataFrame进行查询分析。分析的目标如：

1. 计算所有订单中每年的销售单数，销售总额
2. 计算所有订单每年最大金额订单的销售额
3. 计算所有订单中每年最畅销货品

实现思路及步骤

1. 将数据上传到HDFS
2. 使用SparkContext读取HDFS上的数据
3. 将2中得到的RDD数据转为DataFrame，并注册成为临时表
4. 针对分析目标1，使用join连接3个临时表进行查询
5. 针对分析目标2，首先求出每份订单的销售额以其发生时间，然后以第1步的查询结果和tbData连接，求出每年最大金额订单的销售订单的销售额
6. 针对分析目标3，首先求出每年每个货品的销售利息额，然后求出每年单品销售的最大金额，最后求出每年与销售额最大相符的货品就是最畅销货品

作业要求

1. 提供hdfs上/user/myname/sparkSql/train5data目录下文件列表的截图，应包含文件tbStock.txt，tbStockDetail.txt，tbDate.txt
2. 提供分析目标1,2,3的运行过程及运行结果的截图

实现参考

上传测试数据

```
cd /root/spark
wget -c http://bigdata.hddly.cn/b46488/file/train5data.tar.gz
tar -xzvf ./train5data.tar.gz
hdfs dfs -mkdir -p /user/myname/sparkSql
hdfs dfs -put -d ./train5data /user/myname/sparkSql/
hdfs dfs -ls /user/myname/sparkSql/train5data
```

加载数据

打开spark-shell，在命令行中执行如下：

```
case class tbStock(ordernumber:String,locationid:String,dateid:String) extends Serializable
val tbStockRdd = spark.sparkContext.textFile("/user/myname/sparkSql/train5data/tbStock.txt")
val tbStockDS = tbStockRdd.map(_.split(",")).map(attr=> tbStock(attr(0),attr(1),attr(2))).toDS
tbStockDS.show

case class tbStockDetail(ordernumber:String, rownum:Int, itemid:String, number:Int, price:Double, amount:Double) extends Serializable
val tbStockDetailRdd = spark.sparkContext.textFile("/user/myname/sparkSql/train5data/tbStockDetail.txt")
val tbStockDetailDS = tbStockDetailRdd.map(_.split(",")).map(attr=> tbStockDetail(attr(0),attr(1).trim().toInt,attr(2),attr(3).trim().toInt,attr(4).trim().toDouble, attr(5).trim().toDouble)).toDS
tbStockDetailDS.show()
```

```
10
11 case class tbDate(dateid:String, years:Int, theyear:Int, month:Int, day:Int,
   weekday:Int, week:Int, quarter:Int, period:Int, halfmonth:Int) extends
   Serializable
12 val tbDateRdd =
   spark.sparkContext.textFile("/user/myname/sparkSql/train5data/tbDate.txt")
13 val tbDateDS = tbDateRdd.map(_.split(",")).map(attr=>
   tbDate(attr(0),attr(1).trim().toInt,
   attr(2).trim().toInt,attr(3).trim().toInt, attr(4).trim().toInt,
   attr(5).trim().toInt, attr(6).trim().toInt, attr(7).trim().toInt,
   attr(8).trim().toInt, attr(9).trim().toInt)).toDS
14 tbDateDS.show()
```

使用临时表

```
1 tbStockDS.createOrReplaceTempView("tbStock")
2 tbDateDS.createOrReplaceTempView("tbDate")
3 tbStockDetailDS.createOrReplaceTempView("tbStockDetail")
```

实现查询和统计

```
1 //计算所有订单中每年的销售单数、销售总额
2 spark.sql("SELECT c.theyear, COUNT(DISTINCT a.ordernumber), SUM(b.amount)
   FROM tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber JOIN
   tbDate c ON a.dateid = c.dateid GROUP BY c.theyear ORDER BY c.theyear").show
3 //计算所有订单每年最大金额订单的销售额
4 spark.sql("SELECT a.dateid, a.ordernumber, SUM(b.amount) AS SumOfAmount FROM
   tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber GROUP BY
   a.dateid, a.ordernumber").show
5 spark.sql("SELECT theyear, MAX(c.SumOfAmount) AS SumOfAmount FROM (SELECT
   a.dateid, a.ordernumber, SUM(b.amount) AS SumOfAmount FROM tbStock a JOIN
   tbStockDetail b ON a.ordernumber = b.ordernumber GROUP BY a.dateid,
   a.ordernumber ) c JOIN tbDate d ON c.dateid = d.dateid GROUP BY theyear ORDER
   BY theyear DESC").show
6 //计算所有订单中每年最畅销货品
7 spark.sql("SELECT c.theyear, b.itemid, SUM(b.amount) AS SumOfAmount FROM
   tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber JOIN tbDate c
   ON a.dateid = c.dateid GROUP BY c.theyear, b.itemid").show
8
9 spark.sql("SELECT d.theyear, MAX(d.SumOfAmount) AS MaxOfAmount FROM (SELECT
   c.theyear, b.itemid, SUM(b.amount) AS SumOfAmount FROM tbStock a JOIN
   tbStockDetail b ON a.ordernumber = b.ordernumber JOIN tbDate c ON a.dateid =
   c.dateid GROUP BY c.theyear, b.itemid ) d GROUP BY d.theyear").show
10
11 spark.sql("SELECT DISTINCT e.theyear, e.itemid, f.maxofamount FROM (SELECT
   c.theyear, b.itemid, SUM(b.amount) AS sumofamount FROM tbStock a JOIN
   tbStockDetail b ON a.ordernumber = b.ordernumber JOIN tbDate c ON a.dateid =
   c.dateid GROUP BY c.theyear, b.itemid ) e JOIN (SELECT d.theyear,
   MAX(d.sumofamount) AS maxofamount FROM (SELECT c.theyear, b.itemid,
   SUM(b.amount) AS sumofamount FROM tbStock a JOIN tbStockDetail b ON
   a.ordernumber = b.ordernumber JOIN tbDate c ON a.dateid = c.dateid GROUP BY
   c.theyear, b.itemid ) d GROUP BY d.theyear ) f ON e.theyear = f.theyear AND
   e.sumofamount = f.maxofamount ORDER BY e.theyear").show
```

CH05实训2：基于DataFrame实现老师教学质量统计分析

训练要点

1. 熟悉spark SQL的配置方法
2. 掌握在spark SQL中操作Hive表
3. 掌握创建DataFrame的方法。
4. 掌握DataFrame的查询操作
5. 掌握DataFrame的输出操作

需求说明

在某学校的一个微型班级中，有4个表记录着老师与学生的情况，利用大数据技术进行一定处理，方便老师分析自身的教学质量及学生的学习情况，分别是student表、score 表、course表和teacher表。
student表有4个数据字段，分别为s_id、s_name、s_birth和s_sex, 字段说明如表5一9所示

字段名称	说明
s_id	学生学号
s_name	学生姓名
s_birth	出生日期
s_sex	学生性别

score表有3个数据字段，分别为s_id、c_id和s_score,字段说明如表

字段名称	说明
s_id	学生学号
c_id	课程编号
s_score	分数

course表有3个数据字段，分别表示c_id、c_name和t_id,字段说明如

字段名称	说明
c_id	课程编号
c_name	课程名称
t_id	教师工号

teacher表有2个数据字段，分别为t_id和t_name，字段说明如

字段名称	说明
t_id	教师工号
t_name	教师姓名

为了分析老师的教学质量，请根据score表、course表和teacher表的数据，查询不同老师所教不同课程的平均分.

实现思路及步骤

1. 检查是否已配置spark SQL，若没有，则先配置spark SQL
2. 启动spark-shell、启动Hive元数据服务、访问Hive。
3. 在Hive中创建student数据库，并在student数据库下创建score表、course表和 teacher表，并导人数据
4. 使用spark· sql()方法分别读取Hive中的score表、course表和teacher表的数据.
5. 使用groupBy()、withColumn()、count()、sum()、join()、drop()和cast()方法，按教师工号和课程进行分组，聚合查询不同老师所教不同课程的平均分!
6. 将查询结果输出到Hive的student数据库中的teacher_courseAvg表中

作业要求

1. 提供hdfs上/user/myname/目录下文件列表的截图，应包含文件student.txt，score.txt，course.txt，teacher.txt
2. 提供在Hive中创建student数据库，并在student数据库下创建score表、course表和 teacher表，并导人数据的过程截图
3. 提供分析运行过程及运行结果的截图

实现参考

测试数据文件准备

在master上操作数据文件

```
cd /root/spark
rm -f spark_t_data.tar.gz*
wget https://biglab.site//b59510spark/file/spark_t_data.tar.gz
tar -xzvf ./spark_t_data.tar.gz
hdfs dfs -rm /user/myname/score.txt
hdfs dfs -rm /user/myname/course.txt
hdfs dfs -rm /user/myname/student.txt
hdfs dfs -rm /user/myname/teacher.txt
hdfs dfs -put ./spark_t_data/score.txt /user/myname/
hdfs dfs -put ./spark_t_data/course.txt /user/myname/
hdfs dfs -put ./spark_t_data/student.txt /user/myname/
hdfs dfs -put ./spark_t_data/teacher.txt /user/myname/
```

Hive数据库准备

在master主机上的hive上执行

```
--创建student数据库
create database student;
```

```
--切换student数据库
use student;
--创建score表
create table score(
s_id String,
c_id String,
s_score Int
)
row format delimited fields terminated by ',';
load data local inpath '/root/spark/spark_t_data/score.txt' into table score;
--创建teacher表
create table teacher(
t_id String,
t_name String
)
row format delimited fields terminated by ',';
load data local inpath '/root/spark/spark_t_data/teacher.txt' into table
teacher;

--创建course表
create table course(
c_id String,
c_name String,
t_id String
)
row format delimited fields terminated by ',';
load data local inpath '/root/spark/spark_t_data/course.txt' into table
course;

--创建student表
create table student(
s_id String,
s_name String,
s_birth String,
s_sex String
)
row format delimited fields terminated by ',';
load data local inpath '/root/spark/spark_t_data/student.txt' into table
student;
```

Hive运行结果

```
hive> --创建student数据库
hive> create database student;
116548 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.ql.session.SessionState  - METASTORE_FILTER_HOOK will
be ignored, since hive.security.authorization.manager is set to instance of
HiveAuthorizerFactory.
116557 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
```

```
116589 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
116608 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  - Failed to get database
hive.student, returning NoSuchObjectException
OK
Time taken: 0.429 seconds
hive> --切换student数据库
hive> use student;
OK
Time taken: 0.048 seconds
hive> --创建score表
hive> create table score(
    > s_id String,
    > c_id String,
    > s_score Int
    > )
    > row format delimited fields terminated by ',';
OK
Time taken: 1.064 seconds
hive> load data local inpath '/root/spark/spark_t_data/score.txt' into table
score;
Loading data to table student.score
542020 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
542761 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
OK
Time taken: 1.178 seconds
hive> --创建teacher表
hive> create table teacher(
    > t_id String,
    > t_name String
    > )
    > row format delimited fields terminated by ',';
OK
Time taken: 0.129 seconds
hive> load data local inpath '/root/spark/spark_t_data/teacher.txt' into
table teacher;
Loading data to table student.teacher
700252 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
700499 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
OK
```

```
Time taken: 0.417 seconds
hive>
    > --创建course表
    > create table course(
    > c_id String,
    > c_name String,
    > t_id String
    > )
    > row format delimited fields terminated by ',';
OK
Time taken: 0.099 seconds
hive> load data local inpath '/root/spark/spark_t_data/course.txt' into table
course;
Loading data to table student.course
700797 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
701052 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
OK
Time taken: 0.462 seconds
hive>
    > --创建student表
    > create table student(
    > s_id String,
    > s_name String,
    > s_birth String,
    > s_sex String
    > )
    > row format delimited fields terminated by ',';
OK
Time taken: 0.101 seconds
hive> load data local inpath '/root/spark/spark_t_data/student.txt' into
table student;
Loading data to table student.student
702611 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
702844 [f960459d-ecbf-4bdc-a460-b76bcca31823 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null . Setting
it to value: ignored
OK
Time taken: 0.414 seconds
hive>
```

使用exit;退出hive环境。

Spark脚本参考

在spark-shell环境下运行

```
val teacher = spark.sql("select * from student.teacher")
val score = spark.sql("select * from student.score")
val course = spark.sql("select * from student.course")
// 查询不同老师所教不同课程平均分
val df1 =
score.groupBy("c_id").sum().withColumnRenamed("sum(s_score)","sums")
val df2 = score.groupBy("c_id").count()
val df3 = df1.join(df2,"c_id")
val df4=df3.withColumn("avg", df3.col("sums")/df3.col("count").cast("int"))
val df5 =
df4.join(course,"c_id").join(teacher,"t_id").drop("sums","count","t_id")
df5.show
df5.write.mode(org.apache.spark.sql.SaveMode.Overwrite).saveAsTable("teacher_
courseAvg")
val rlt = spark.sql("select * from teacher_courseAvg")
rlt.show
```

Spark运行结果

```
scala> val teacher = spark.sql("select * from student.teacher")
teacher: org.apache.spark.sql.DataFrame = [t_id: string, t_name: string]

scala> val score = spark.sql("select * from student.score")
score: org.apache.spark.sql.DataFrame = [s_id: string, c_id: string ... 1
more field]

scala> val course = spark.sql("select * from student.course")
course: org.apache.spark.sql.DataFrame = [c_id: string, c_name: string ... 1
more field]

scala> // 查询不同老师所教不同课程平均分

scala> val df1 =
score.groupBy("c_id").sum().withColumnRenamed("sum(s_score)","sums")
df1: org.apache.spark.sql.DataFrame = [c_id: string, sums: bigint]

scala> val df2 = score.groupBy("c_id").count()
df2: org.apache.spark.sql.DataFrame = [c_id: string, count: bigint]

scala> val df3 = df1.join(df2,"c_id")
df3: org.apache.spark.sql.DataFrame = [c_id: string, sums: bigint ... 1 more
field]

scala> val df4=df3.withColumn("avg",
df3.col("sums")/df3.col("count").cast("int"))
df4: org.apache.spark.sql.DataFrame = [c_id: string, sums: bigint ... 2 more
fields]

scala> val df5 =
df4.join(course,"c_id").join(teacher,"t_id").drop("sums","count","t_id")
```

```
df5: org.apache.spark.sql.DataFrame = [c_id: string, avg: double ... 2 more fields]

scala> df5.show
+----+-----------------+------+------+

|c_id|              avg|c_name|t_name|
+----+-----------------+------+------+
|  01|             64.5|  语文|  李四|
|  03|             68.5|  英语|  王五|
|  02|72.66666666666667|  数学|  张三|
+----+-----------------+------+------+


scala> df5.write.mode(org.apache.spark.sql.SaveMode.Overwrite).saveAsTable("teacher_courseAvg")

scala> val rlt = spark.sql("select * from teacher_courseAvg")
rlt: org.apache.spark.sql.DataFrame = [c_id: string, avg: double ... 2 more fields]

scala> rlt.show
+----+-----------------+------+------+

|c_id|              avg|c_name|t_name|
+----+-----------------+------+------+
|  01|             64.5|  语文|  李四|
|  03|             68.5|  英语|  王五|
|  02|72.66666666666667|  数学|  张三|
+----+-----------------+------+------+


scala>

```

CH05实训3：基于DataFrame实现学生成绩统计分析

训练要点

1. 掌握在spark SQL中操作Hive表。
2. 掌握创建DataFrame对象的方法
3. 掌握DataFrame的查询操作

需求说明

为了进行学生成绩统计分析，需要根据实训1中student表、course表和score表这3 个表的数据，完成以下查询任务。

1. 查询所有学生的学生学号、学生姓名、选课总数和所有课程的总成绩；

2. 查询课程名称为“数学"，且分数低于60的学生姓名和分数
3. 查询学生的平均成绩及名次

实现思路及步骤

1. 在Hive的student数据库中创建student表并导人数据，因为score表和course表在实训1中已经创建过，所以无须再创建
2. 使用spark· sql()方法分别读取Hive中的score表、course表和student表的数据
3. 使用groupBy()、sum()、count()、join()和drop()方法，查询所有学生的学生学号、学生姓名、选课总数和所有课程的总成绩。
4. 使用join()和where()方法，查询课程名称为“数学"，且分数低于60的学生姓名和分数
5. 使用groupBy()、join()、sum()、count()、withColumn()、col()、selectExpr()、drop() 和cast()方法，查询学生的平均成绩及名次

作业要求

1. 提供hdfs上/user/myname/sparkSql/train5data目录下文件列表的截图，应包含文件student.txt，score.txt，course.txt，teacher.txt
2. 提供在Hive中创建student数据库，并在student数据库下创建score表、course表和 teacher表，并导人数据的过程截图
3. 提供分析步骤4，5的运行过程及运行结果的截图
4. 提供hdft上user/myname/sparkSql/student_courseAvg结果表的内容的截图

实现参考

测试数据准备

参考一，如果已完成测试数据准备则不用再执行。

脚本参考

启动spark-shell

[root@spark2-master-0 sbin]# spark-shell

```
// 读取数据
val student = spark.sql("select * from student.student")
val score = spark.sql("select * from student.score")
val course = spark.sql("select * from student.course")
// 查询所有同学的学生编号、学生姓名、选课总数、所有课程的总成绩
val df1 = score.groupBy("s_id").sum("s_score")
//val df2 = score.groupBy("s_id").count().join(df1,"s_id","left") //error
val df2 = score.groupBy("s_id").count().join(df1,Seq("s_id"),"left_outer")
//val df3 = student.drop("s_birth","s_sex").join(df2,"s_id","left") //error
val df3 = student.drop("s_birth","s_sex").join(df2,Seq("s_id"),"left_outer")
df3.show
//查询课程名称为"数学"，且分数低于60的学生姓名和分数
val df4 = score.join(course, "c_id").where("c_name='数学' and s_score<60")
df4.show
// 查询学生平均成绩及其名次
val df5 =
score.groupBy("s_id").sum("s_score").join(score.groupBy("s_id").count(),"s_id
")
```

```
val df6 = df5.withColumn("avg",(df5.col("sum(s_score)")/df5.col("count")).cast("int")).drop("sum(s_score)","count")
val df7 = df6.selectExpr("s_id","avg","rank() over(order by avg desc) as ord")
df7.show
df7.write.mode(org.apache.spark.sql.SaveMode.Overwrite).saveAsTable("student_courseAvg")
val rlt = spark.sql("select * from student_courseAvg")
rlt.show
```

运行过程结果

```
scala> // 读取数据

scala> val student = spark.sql("select * from student.student")
student: org.apache.spark.sql.DataFrame = [s_id: string, s_name: string ... 2 more fields]

scala> val score = spark.sql("select * from student.score")
score: org.apache.spark.sql.DataFrame = [s_id: string, c_id: string ... 1 more field]

scala> val course = spark.sql("select * from student.course")
course: org.apache.spark.sql.DataFrame = [c_id: string, c_name: string ... 1 more field]

scala> // 查询所有同学的学生编号、学生姓名、选课总数、所有课程的总成绩

scala> val df1 = score.groupBy("s_id").sum("s_score")
df1: org.apache.spark.sql.DataFrame = [s_id: string, sum(s_score): bigint]

scala> //val df2 = score.groupBy("s_id").count().join(df1,"s_id","left") //error

scala> val df2 = score.groupBy("s_id").count().join(df1,Seq("s_id"),"left_outer")
df2: org.apache.spark.sql.DataFrame = [s_id: string, count: bigint ... 1 more field]

scala> //val df3 = student.drop("s_birth","s_sex").join(df2,"s_id","left") //error

scala> val df3 = student.drop("s_birth","s_sex").join(df2,Seq("s_id"),"left_outer")
df3: org.apache.spark.sql.DataFrame = [s_id: string, s_name: string ... 2 more fields]

scala> df3.show
+----+------+-----+------------+

|s_id|s_name|count|sum(s_score)|
+----+------+-----+------------+
|  01|  赵雷|    3|         269|
|  02|  钱电|    3|         210|
```

```
|  03|  孙风|    3|         240|
|  04|  李云|    3|         100|
|  05|  周梅|    2|         163|
|  06|  吴兰|    2|          65|
|  07|  郑竹|    2|         187|
|  08|  王菊| null|        null|
+----+------+-----+------------+


scala> //查询课程名称为"数学"，且分数低于60的学生姓名和分数

scala> val df4 = score.join(course, "c_id").where("c_name='数学' and s_score<60")
df4: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [c_id: string, s_id: string ... 3 more fields]

scala> df4.show
+----+----+-------+------+----+
|c_id|s_id|s_score|c_name|t_id|
+----+----+-------+------+----+
|  02|  04|     30|  数学|  01|
+----+----+-------+------+----+


scala> // 查询学生平均成绩及其名次

scala> val df5 = score.groupBy("s_id").sum("s_score").join(score.groupBy("s_id").count(),"s_id")
df5: org.apache.spark.sql.DataFrame = [s_id: string, sum(s_score): bigint ... 1 more field]

scala> val df6 = df5.withColumn("avg", (df5.col("sum(s_score)")/df5.col("count")).cast("int")).drop("sum(s_score)","count")
df6: org.apache.spark.sql.DataFrame = [s_id: string, avg: int]

scala> val df7 = df6.selectExpr("s_id","avg","rank() over(order by avg desc) as ord")
df7: org.apache.spark.sql.DataFrame = [s_id: string, avg: int ... 1 more field]

scala> df7.show
+----+---+---+

|s_id|avg|ord|
+----+---+---+
|  07| 93|  1|
|  01| 89|  2|
|  05| 81|  3|
|  03| 80|  4|
|  02| 70|  5|
|  04| 33|  6|
|  06| 32|  7|
+----+---+---+
```

```
78
79
80  scala>
    df7.write.mode(org.apache.spark.sql.SaveMode.Overwrite).saveAsTable("student_
    courseAvg")
81
82  scala> val rlt = spark.sql("select * from student_courseAvg")
83  rlt: org.apache.spark.sql.DataFrame = [s_id: string, avg: int ... 1 more
    field]
84
85  scala> rlt.show
86  +----+---+---+
87  |s_id|avg|ord|
88  +----+---+---+
89  |  07| 93|  1|
90  |  01| 89|  2|
91  |  05| 81|  3|
92  |  03| 80|  4|
93  |  02| 70|  5|
94  |  04| 33|  6|
95  |  06| 32|  7|
96  +----+---+---+
97
98
99  scala>
```