

《大数据综合实训》练习01

• 说明

- 在D盘创建考生文件夹：命名为“《大数据综合实训练习2班》”
- 在IDEA的项目中，创建Spark任务，名称为 `test01`
- 答题结束后，需要提交以下资料
 - 1.需在IDEA中导出 `test01.scala` 文件为 `HTML` 文件，并保存在考生文件夹
 - 2.程序运行结束后，需对整个IDEA的界面进行截图【截图界面包括运行的结果和项目名称】，命名“`idea.jpg`”保存在考生文件夹，截图需要能看到“项目、任务名称+运行结果”
 - 3.将考生文件夹打包提交。

一. 启动环境

```
# 1.1
能启动VMware和FinalShell连接虚拟机即可得分

# 1.2 启动hadoop集群
start-dfs.sh
start-yarn.sh

# 1.3 启动Hive服务
nohup hive --service metastore &
nohup hive --service hiveserver2 &
```

二. 加载数据到HDFS

```
-- 2.1 在HDFS的根目录下创建名为 `exam_data01` 的文件夹
注意：如果此目录已存在，先删除

-- 2.2 加载examdata.csv数据文件到刚才创建的 `exam_data01` 的文件夹中
```

三. 加载数据到Hive

接收Jar包后，到集群运行加载数据

```
[root@master ~]# spark-submit --master yarn --deploy-mode client \
--class org.example.LoadData Spark2024-1.0.jar
```

四. 数据清洗和指标运算

```
package org.example

// 导入必要的Spark SQL库
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

// 定义一个名为test01的对象，它是这个作业的主体
object test01 {
  // main函数是程序的入口点
  def main(args: Array[String]): Unit = {
    // 初始化Spark会话
    val spark = SparkSession.builder()
      .appName("DWD Data Processor") // 设置应用程序名称
      .master("local[*]") // 设置运行模式为本地模式，使用所有可用的核心
      .enableHiveSupport() // 启用对Hive的支持，允许与Hive交互
      .config("hive.metastore.uris", "thrift://192.168.36.100:9083") // 设置Hive元
数据存储在Thrift服务地址
      .config("spark.sql.warehouse.dir",
"hdfs://192.168.36.100:9000/user/hive/warehouse") // 设置Hive仓库在HDFS上的位置
      .getOrCreate() // 创建Spark会话，如果存在则获取现有的

    // 调用自定义的cleanData函数，清洗和指标运算
    cleanData(spark)

    // 执行完毕后关闭Spark会话
    // spark.stop()
  }

  // 定义一个处理数据并将其存储到DWD层的函数
  def cleanData(spark: SparkSession): Unit = {
    // 从ODS层读取用户和订单数据
    val odsSalesDF = spark.table("ods.ods_sales")

    // 对用户数据进行清洗：删除空值、重复值
    val cleanedSalesDF = odsSalesDF
      .na.drop() // 删除所有列中的空值
      .dropDuplicates() // 删除重复值
    println("完成数据进行清洗：删除空值、重复值.....")

    // 计算销售总额（总金额 = 单价 * 销售数量 * (1 - 折扣)），并添加为新列total_sales
    val dwdSalesDF = cleanedSalesDF.withColumn(
      "total_sales", col("price") * col("quantity") * (lit(1) - col("discount"))
    )

    println("dwdSalesDF数据表第1行的内容")
    dwdSalesDF.show(1)

    println("程序运行结果：")
    // 1. 汇总所有产品的销售总额
    val totalSalesAmountDF =
dwdSalesDF.agg(sum("total_sales").alias("total_sales_amount"))
    val totalSalesAmount = totalSalesAmountDF.first().getDouble(0)
    println(s"1.所有产品的销售总额：$totalSalesAmount")
  }
}
```

```

// 2. 计算 "家用电器" 的销售总额
val SalesDF = dwdSalesDF.filter(col("product_category") === "家用电器")
val TotalSales = SalesDF.agg(sum("total_sales").alias("total_sales"))
val SalesAmount = TotalSales.first().getDouble(0)
println(s"2.家用电器的销售总额: $SalesAmount")

// 3. 查询 "总销售额" 最低的销售人员的姓名
val topSalesPersonDF = dwdSalesDF`
    .groupBy("salesperson") // 按销售人员姓名分组
    .agg(sum("total_sales").alias("total_sales")) // 计算每个销售人员的总销售额
    .orderBy(asc("total_sales")) // 按总销售额升序排列
    .limit(1) // 取总销售额最高的销售人员
val topSalesPersonName = topSalesPersonDF.first().getString(0) // 获取销售人员的姓名
println(s"3.总销售额最低的销售人员: $topSalesPersonName")

// 4. 查询销售额高于 10000 元的订单数量
val lowSalesOrderCount = dwdSalesDF.filter(col("total_sales") >
10000).count()
println(s"4.销售额高于10000元的订单数量: $lowSalesOrderCount")

}
}

```