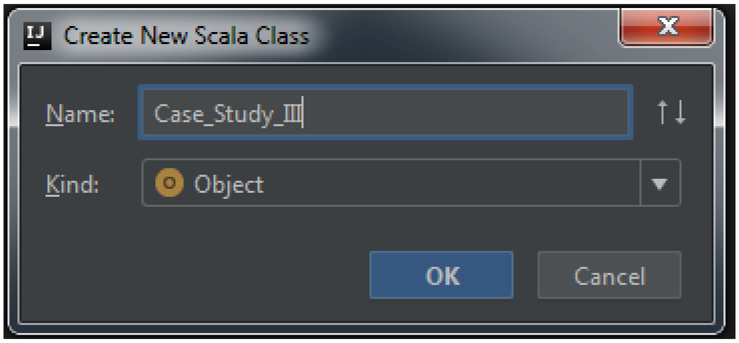
Case Study III for Session 20 – Spark SQL

- Prachi Mohite

In this Assignment we will be using IDEA IntelliJ to Complete the given Task

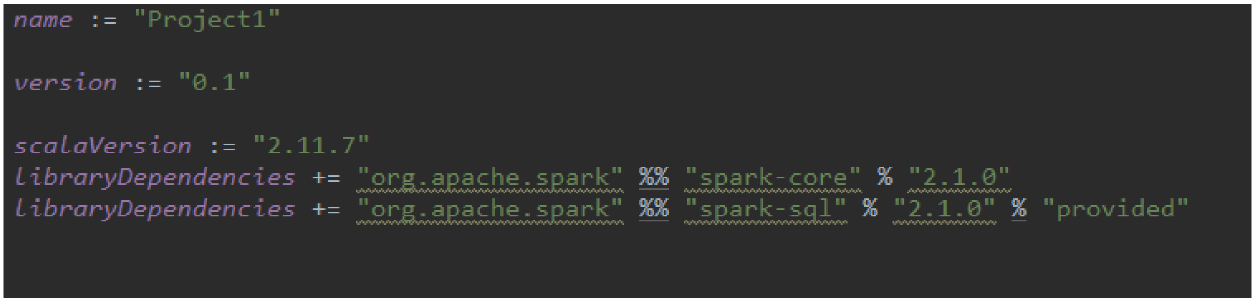
1. Created new Project and added scala object named as Case_Study_III as below



Name: Case_Study_III

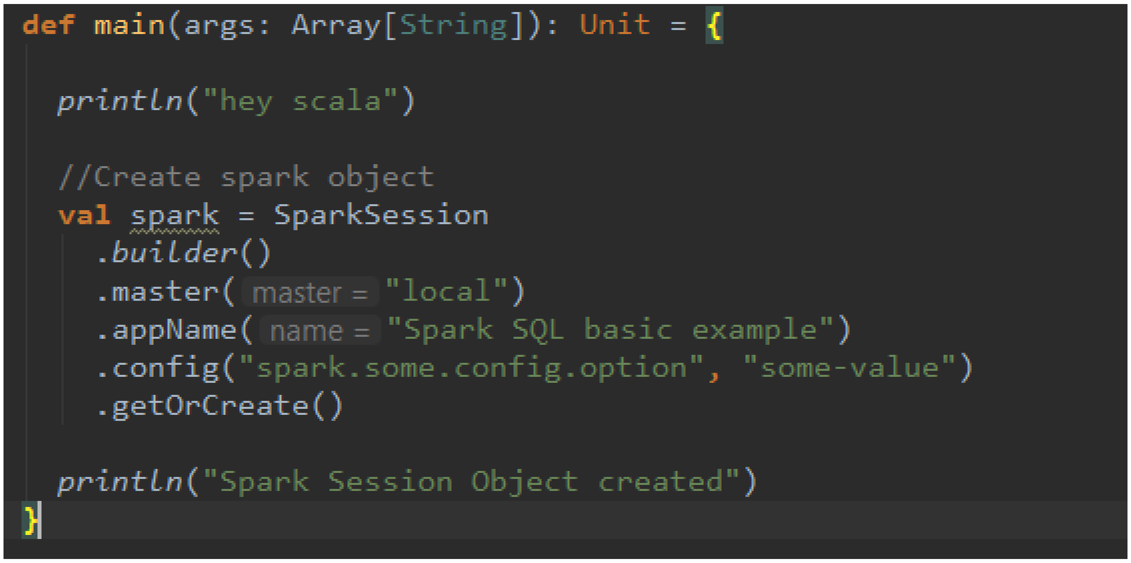
Kind: Object

2. To add the required dependencies we have created scala sbt project in IDEA and added library dependency from maven repository as below



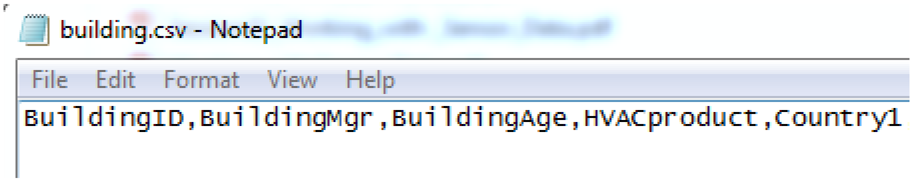
```
name := "Project1"

version := "0.1"

scalaVersion := "2.11.7"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.1.0" % "provided"
```

3. Added main function and created the spark object as below



```
def main(args: Array[String]): Unit = {

  println("hey scala")

  //Create spark object
  val spark = SparkSession
    .builder()
    .master( master = "local")
    .appName( name = "Spark SQL basic example")
    .config("spark.some.config.option", "some-value")
    .getOrCreate()

  println("Spark Session Object created")
}
```

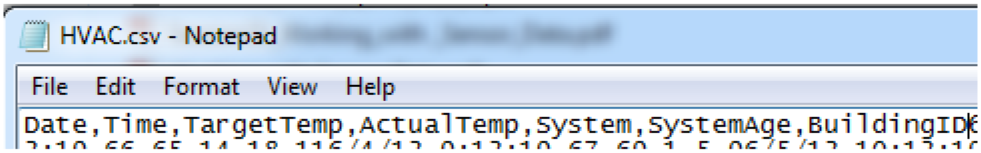
4. We will be using below dataset for this assignment
    a. Building CSV
    b. Columns are – BuildingID, BuildingMgr, BuildingAge, HVACproduct, country



```
BuildingID,BuildingMgr,BuildingAge,HVACproduct,Country1
```

c. HVAC.csv

d. Columns are – Date, Time , TargetTemp, ActualTemp, System, SystemAge, BuildingID



```
Date,Time,TargetTemp,ActualTemp,System,SystemAge,BuildingID
```

Task 1

Load HVAC.csv into temporary table.

Solution Approach –

In this case we have to first create a Dataframe and then register that data frame as a temporary table. This can be achieved by two approaches as below

- **Inferring the Schema using Reflection:** This method uses reflection to generate the schema of an RDD that contains specific types of objects.
  - In this case we have create the case class which will reflect the schema of dataframe / temp table which should hold values from the file.
  - Once case class is created, we have to read the text file and create a RDD based on spit operator specified for that file data. Apply transformations on RDDF and get the required Dataframe created.
  - Create a temporary table from the Data frame.

Note : Above approach we have applied in Assignment 20 , 21.

- **Programmatically Specifying the Schema:** The second method for creating DataFrame is through programmatic interface that allows you to construct a schema and then apply it to an existing RDD
  - In this method we have to create a manual schema which should matches to our requirement and data of file to be loaded.
  - Create a Data frame using createDataFrame method on RowRDD or while loading only apply schema of load method

Note : In this case study we will be using this Approach

1. Create a manual schema which matches data in both the files building.csv and HVAc.csv
    a. While creating the schema we have used Struct type.
    b. StructType is a built-in data type used for Schema definition in Spark SQL, to represent a collection of StructFields that together define a schema or its part.

<schema-name> = new structType<array_of_columns><Struct_field>(<column_name>,<data_type_of_column>,<nullable_or_not_nullable(true/false)>)

2.
```
//Schema for HVAC.csv
val Manual_schema_HVAC = new StructType(Array(new
StructField("Date", StringType,
  true),
  new StructField("Time", StringType, false),
  new StructField("TargetTemp", LongType, true),
  new StructField("ActualTemp", LongType, false),
  new StructField("System", LongType, false),
  new StructField("SystemAge", LongType, false),
  new StructField("BuildingID", LongType, false)))

//Schema for building.csv
  val Manual_schema_Building = new StructType(Array(new
StructField("BuildingID",
  LongType, true),
  new StructField("BuildingMgr", StringType, false),
  new StructField("BuildingAge", LongType, true),
  new StructField("HVACproduct", StringType, false),
  new StructField("Country", StringType, false)))
```

3. Load the the CSV files from local system to Spark as below
    a. Spark SQL provides inbuilt support for only 3 types of data sources
        i. Parquet (This is default)
        ii. JSON
        iii. JDBC
        iv. For CSV we have to use another library spark-csv this was prior to Spark 2.0. As of Spark version 2.0 and up, spark-csv is part of core Spark functionality and doesn't require a separate library.

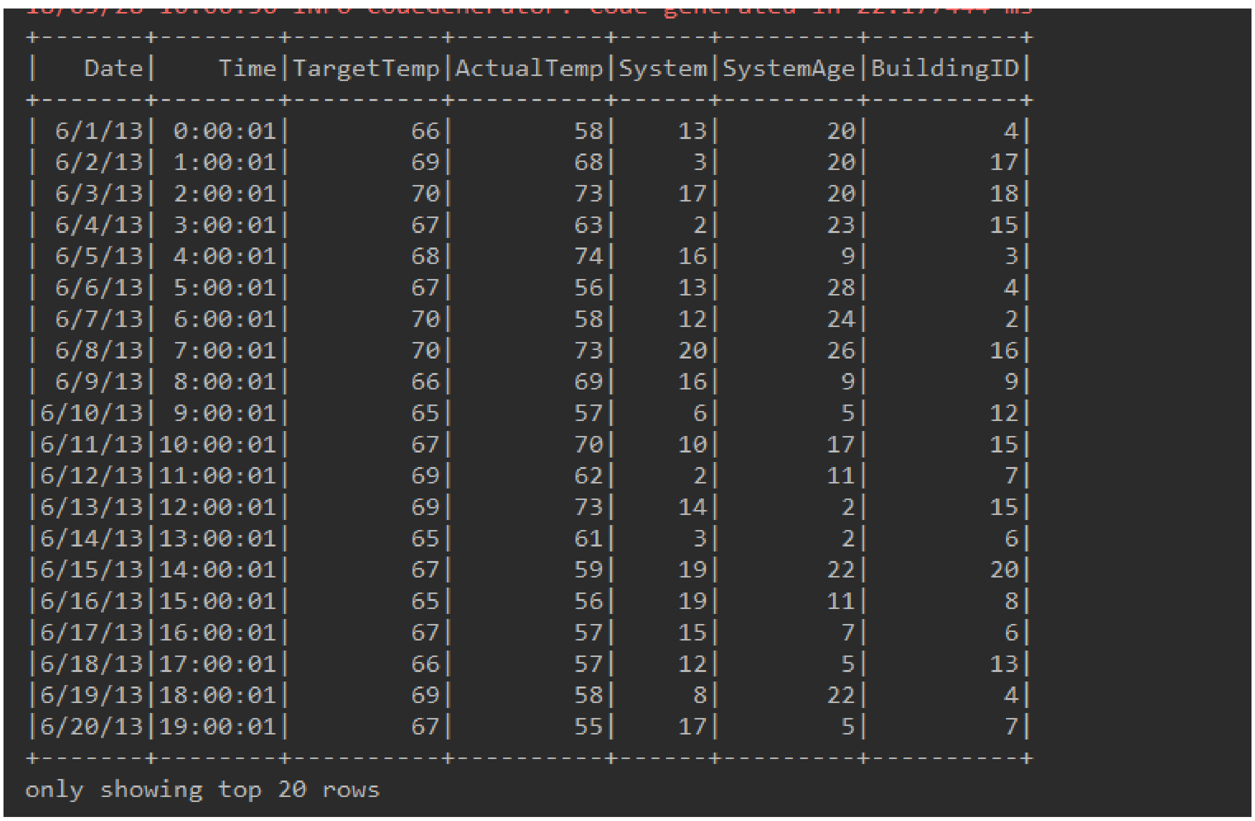
Below code is how to read and load the csv file and create a dataframe

We are using the CSV file read format, this provides various options of which we have used a few of them, which are as follows:

- option to remove the header from the input file.
- given the manual schema that we have created in the previous step
- the path where the CSV file is saved in the local file system.

```
//Reading the HVAC csv file
val HVAC = spark.read.format("CSV")
  .option("header",true)
  .schema(Manual_schema_HVAC)
  .load("E:\\Prachi IMP\\Hadoop\\Case Studies - Assignment\\Case Study III\\HVAC.csv")


//Displaying the dataframe contents
HVAC.show()
```

Output of the show command



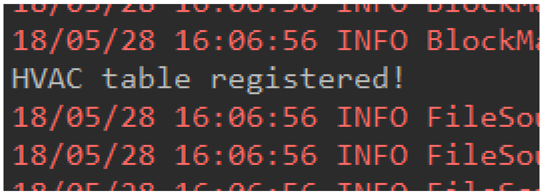
```
+-------+--------+----------+----------+------+---------+----------+
|   Date|    Time|TargetTemp|ActualTemp|System|SystemAge|BuildingID|
+-------+--------+----------+----------+------+---------+----------+
| 6/1/13| 0:00:01|        66|        58|    13|       20|         4|
| 6/2/13| 1:00:01|        69|        68|     3|       20|        17|
| 6/3/13| 2:00:01|        70|        73|    17|       20|        18|
| 6/4/13| 3:00:01|        67|        63|     2|       23|        15|
| 6/5/13| 4:00:01|        68|        74|    16|        9|         3|
| 6/6/13| 5:00:01|        67|        56|    13|       28|         4|
| 6/7/13| 6:00:01|        70|        58|    12|       24|         2|
| 6/8/13| 7:00:01|        70|        73|    20|       26|        16|
| 6/9/13| 8:00:01|        66|        69|    16|        9|         9|
|6/10/13| 9:00:01|        65|        57|     6|        5|        12|
|6/11/13|10:00:01|        67|        70|    10|       17|        15|
|6/12/13|11:00:01|        69|        62|     2|       11|         7|
|6/13/13|12:00:01|        69|        73|    14|        2|        15|
|6/14/13|13:00:01|        65|        61|     3|        2|         6|
|6/15/13|14:00:01|        67|        59|    19|       22|        20|
|6/16/13|15:00:01|        65|        56|    19|       11|         8|
|6/17/13|16:00:01|        67|        57|    15|        7|         6|
|6/18/13|17:00:01|        66|        57|    12|        5|        13|
|6/19/13|18:00:01|        69|        58|     8|       22|         4|
|6/20/13|19:00:01|        67|        55|    17|        5|         7|
+-------+--------+----------+----------+------+---------+----------+
only showing top 20 rows
```

4. Registering the temporary table with registerTempTable command

a. registerTempTable() creates an in-memory table that is scoped to the cluster in which it was created. The data is stored using Hive's highly-optimized, in-memory columnar format.
b. This is important for dashboards as dashboards running in a different cluster (ie. the single Dashboard Cluster) will not have access to the temp tables registered in another cluster.
c. Re-registering a temp table of the same name (using overwrite=true) but with new data causes an atomic memory pointer switch so the new data is seemlessly updated and immediately accessble for querying (ie. from a Dashboard).

```
//Registering the temporary table
HVAC.registerTempTable("HVAC_table")
println("HVAC table registered!")
```

Output



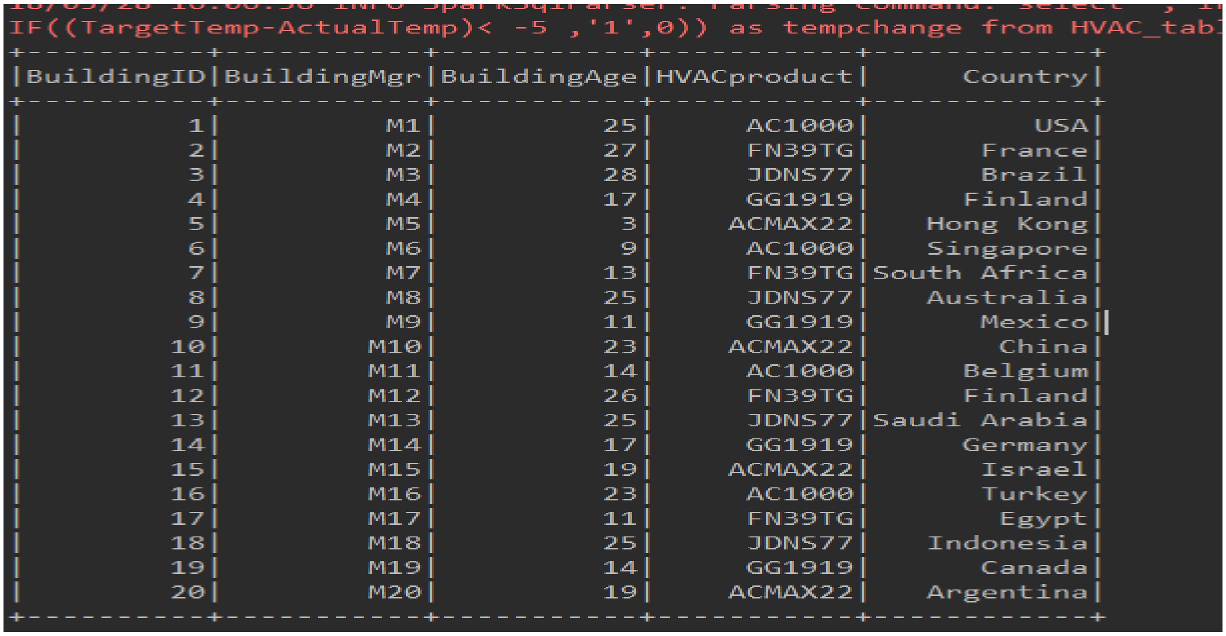
```
18/05/28 16:06:56 INFO BlockMa
HVAC table registered!
18/05/28 16:06:56 INFO FileSou
18/05/28 16:06:56 INFO FileSou
```

5. Same above steps are followed to create the dataframe for building.csv file and create a temp table

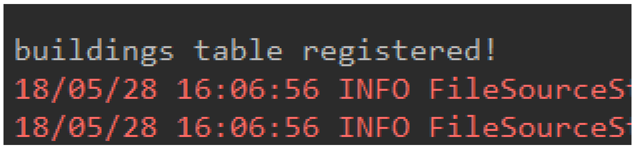
Code

```
//Reading the building.csv file and creating the temp table
val buildings = spark.read.format("CSV")
  .option("header", true)
  .schema(Manual_schema_Building)
  .load("E:\\Prachi IMP\\Hadoop\\Case Studies - Assignment\\Case Study III\\building.csv")
buildings.show()
buildings.registerTempTable("building_table")
println("buildings table registered!")
```

Output



```
18/05/28 16:06:56 INFO SparkSqlParser: Parsing command: select *, IF
IF((TargetTemp-ActualTemp)< -5 ,'1',0)) as tempchange from HVAC_tab
+----------+-----------+-----------+-----------+------------+
|BuildingID|BuildingMgr|BuildingAge|HVACproduct|     Country|
+----------+-----------+-----------+-----------+------------+
|         1|         M1|         25|     AC1000|         USA|
|         2|         M2|         27|     FN39TG|      France|
|         3|         M3|         28|     JDNS77|      Brazil|
|         4|         M4|         17|     GG1919|     Finland|
|         5|         M5|          3|    ACMAX22|   Hong Kong|
|         6|         M6|          9|     AC1000|   Singapore|
|         7|         M7|         13|     FN39TG|South Africa|
|         8|         M8|         25|     JDNS77|   Australia|
|         9|         M9|         11|     GG1919|      Mexico||
|        10|        M10|         23|    ACMAX22|       China|
|        11|        M11|         14|     AC1000|     Belgium|
|        12|        M12|         26|     FN39TG|     Finland|
|        13|        M13|         25|     JDNS77|Saudi Arabia|
|        14|        M14|         17|     GG1919|     Germany|
|        15|        M15|         19|    ACMAX22|      Israel|
|        16|        M16|         23|     AC1000|      Turkey|
|        17|        M17|         11|     FN39TG|       Egypt|
|        18|        M18|         25|     JDNS77|   Indonesia|
|        19|        M19|         14|     GG1919|      Canada|
|        20|        M20|         19|    ACMAX22|   Argentina|
+----------+-----------+-----------+-----------+------------+
```



```
buildings table registered!
18/05/28 16:06:56 INFO FileSourceS
18/05/28 16:06:56 INFO FileSourceS
```

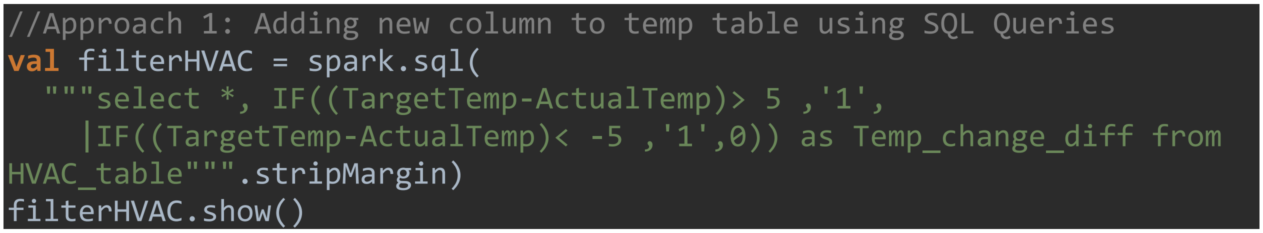
Task 1.2

Add a new column, tempchange - set to 1, if there is a change of greater than +/-5 between actual and target temperature

Solution Approach –

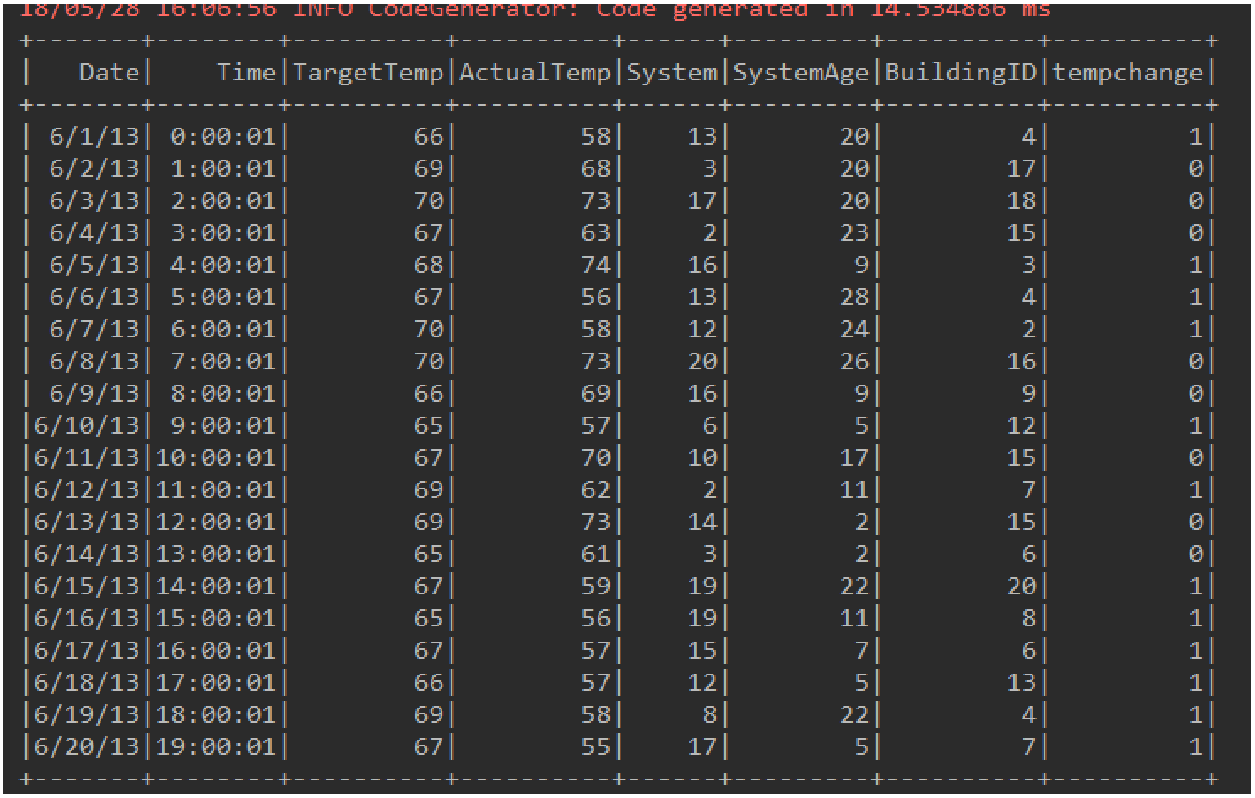
AS we have Temp Table and Dataframe created, a new column can be added with below two approaches

Approach 1: Using SQL queries on Temp Table



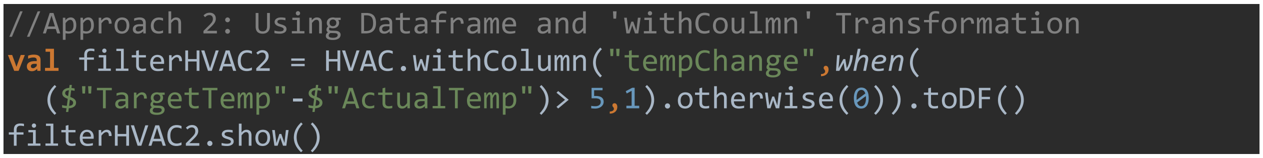
```
//Approach 1: Adding new column to temp table using SQL Queries
val filterHVAC = spark.sql(
  """select *, IF((TargetTemp-ActualTemp)> 5 ,'1',
    |IF((TargetTemp-ActualTemp)< -5 ,'1',0)) as Temp_change_diff from
HVAC_table""".stripMargin)
filterHVAC.show()
```

Output



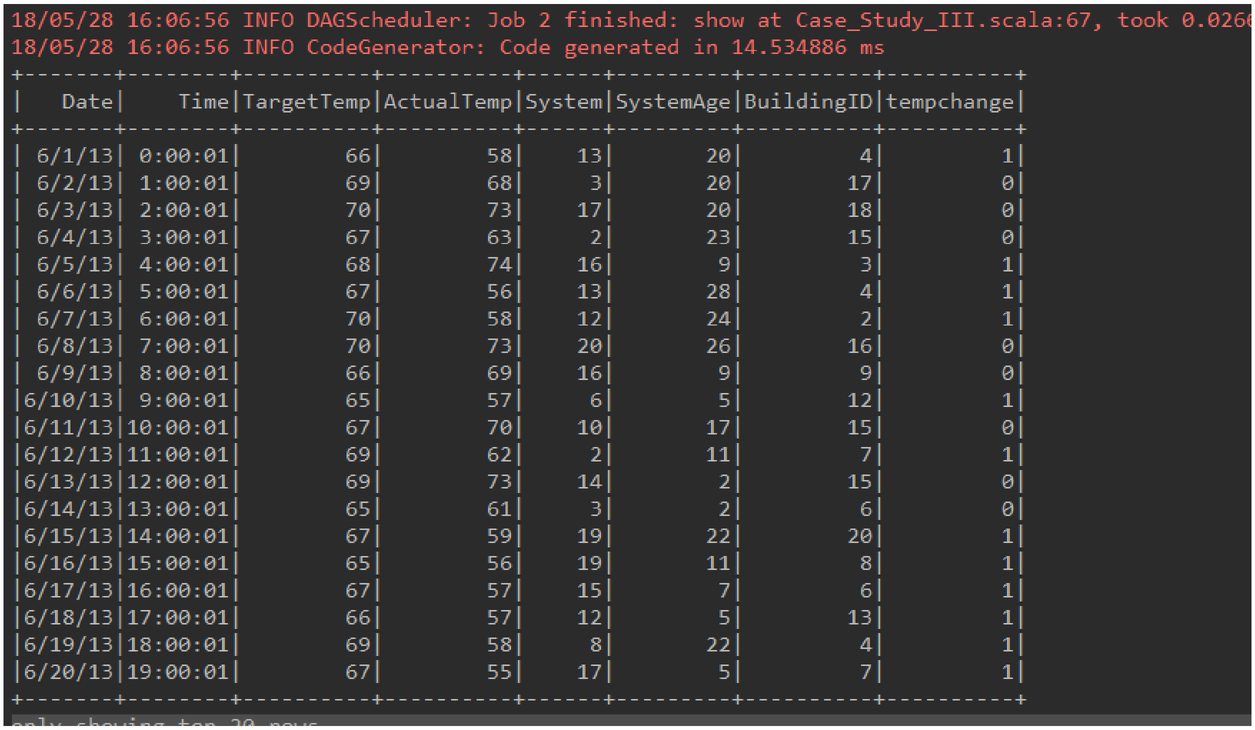
```
18/05/28 16:06:56 INFO CodeGenerator: Code generated in 14.534886 ms
+-------+--------+----------+----------+------+---------+----------+----------+
|   Date|    Time|TargetTemp|ActualTemp|System|SystemAge|BuildingID|tempchange|
+-------+--------+----------+----------+------+---------+----------+----------+
| 6/1/13| 0:00:01|        66|        58|    13|       20|         4|         1|
| 6/2/13| 1:00:01|        69|        68|     3|       20|        17|         0|
| 6/3/13| 2:00:01|        70|        73|    17|       20|        18|         0|
| 6/4/13| 3:00:01|        67|        63|     2|       23|        15|         0|
| 6/5/13| 4:00:01|        68|        74|    16|        9|         3|         1|
| 6/6/13| 5:00:01|        67|        56|    13|       28|         4|         1|
| 6/7/13| 6:00:01|        70|        58|    12|       24|         2|         1|
| 6/8/13| 7:00:01|        70|        73|    20|       26|        16|         0|
| 6/9/13| 8:00:01|        66|        69|    16|        9|         9|         0|
|6/10/13| 9:00:01|        65|        57|     6|        5|        12|         1|
|6/11/13|10:00:01|        67|        70|    10|       17|        15|         0|
|6/12/13|11:00:01|        69|        62|     2|       11|         7|         1|
|6/13/13|12:00:01|        69|        73|    14|        2|        15|         0|
|6/14/13|13:00:01|        65|        61|     3|        2|         6|         0|
|6/15/13|14:00:01|        67|        59|    19|       22|        20|         1|
|6/16/13|15:00:01|        65|        56|    19|       11|         8|         1|
|6/17/13|16:00:01|        67|        57|    15|        7|         6|         1|
|6/18/13|17:00:01|        66|        57|    12|        5|        13|         1|
|6/19/13|18:00:01|        69|        58|     8|       22|         4|         1|
|6/20/13|19:00:01|        67|        55|    17|        5|         7|         1|
+-------+--------+----------+----------+------+---------+----------+----------+
```

Approach 2: Using Dataframe and '*withcoulmn'* transformations to get new column which is added conditionally



```
//Approach 2: Using Dataframe and 'withCoulmn' Transformation
val filterHVAC2 = HVAC.withColumn("tempChange",when(
  ($"TargetTemp"-$"ActualTemp")> 5,1).otherwise(0)).toDF()
filterHVAC2.show()
```

Output



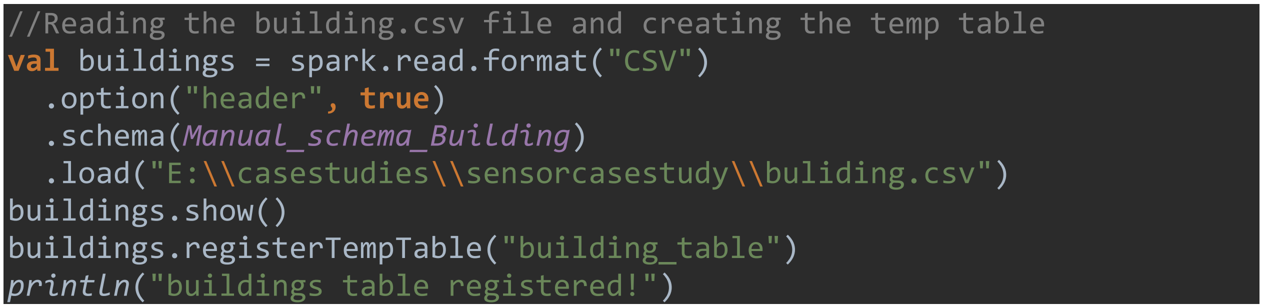
```
18/05/28 16:06:56 INFO DAGScheduler: Job 2 finished: show at Case_Study_III.scala:67, took 0.026
18/05/28 16:06:56 INFO CodeGenerator: Code generated in 14.534886 ms
+-------+--------+----------+----------+------+---------+----------+----------+
|   Date|    Time|TargetTemp|ActualTemp|System|SystemAge|BuildingID|tempchange|
+-------+--------+----------+----------+------+---------+----------+----------+
| 6/1/13| 0:00:01|        66|        58|    13|       20|         4|         1|
| 6/2/13| 1:00:01|        69|        68|     3|       20|        17|         0|
| 6/3/13| 2:00:01|        70|        73|    17|       20|        18|         0|
| 6/4/13| 3:00:01|        67|        63|     2|       23|        15|         0|
| 6/5/13| 4:00:01|        68|        74|    16|        9|         3|         1|
| 6/6/13| 5:00:01|        67|        56|    13|       28|         4|         1|
| 6/7/13| 6:00:01|        70|        58|    12|       24|         2|         1|
| 6/8/13| 7:00:01|        70|        73|    20|       26|        16|         0|
| 6/9/13| 8:00:01|        66|        69|    16|        9|         9|         0|
|6/10/13| 9:00:01|        65|        57|     6|        5|        12|         1|
|6/11/13|10:00:01|        67|        70|    10|       17|        15|         0|
|6/12/13|11:00:01|        69|        62|     2|       11|         7|         1|
|6/13/13|12:00:01|        69|        73|    14|        2|        15|         0|
|6/14/13|13:00:01|        65|        61|     3|        2|         6|         0|
|6/15/13|14:00:01|        67|        59|    19|       22|        20|         1|
|6/16/13|15:00:01|        65|        56|    19|       11|         8|         1|
|6/17/13|16:00:01|        67|        57|    15|        7|         6|         1|
|6/18/13|17:00:01|        66|        57|    12|        5|        13|         1|
|6/19/13|18:00:01|        69|        58|     8|       22|         4|         1|
|6/20/13|19:00:01|        67|        55|    17|        5|         7|         1|
+-------+--------+----------+----------+------+---------+----------+----------+
only showing top 20 rows
```

Note: Instead of writing the conditions for adding new column inline in SQL Query or in withCoulmn transformation, we can write a udf and register it and we can use it as well

Task 2 Load the building.csv file in temporary table

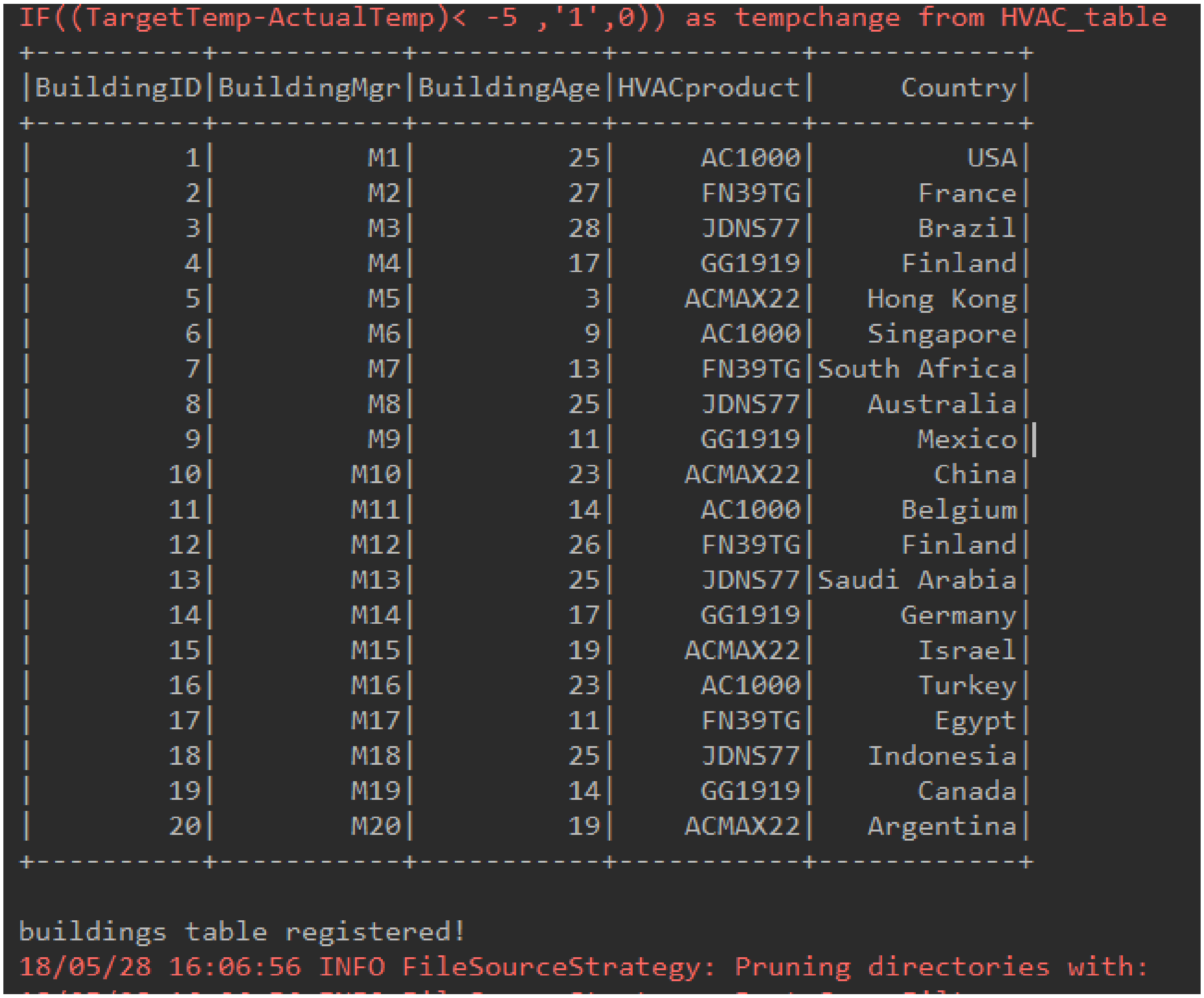
As mentioned above already loaded building.csv as well

Code



```
//Reading the building.csv file and creating the temp table
val buildings = spark.read.format("CSV")
  .option("header", true)
  .schema(Manual_schema_Building)
  .load("E:\\casestudies\\sensorcasestudy\\buliding.csv")
buildings.show()
buildings.registerTempTable("building_table")
println("buildings table registered!")
```

Output



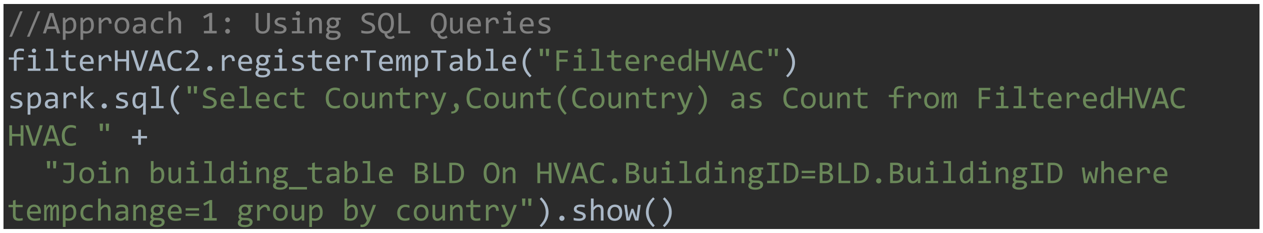
```
IF((TargetTemp-ActualTemp)< -5 ,'1',0)) as tempchange from HVAC_table
+----------+-----------+-----------+----------+------------+
|BuildingID|BuildingMgr|BuildingAge|HVACproduct|     Country|
+----------+-----------+-----------+----------+------------+
|         1|         M1|         25|    AC1000|         USA|
|         2|         M2|         27|    FN39TG|      France|
|         3|         M3|         28|    JDNS77|      Brazil|
|         4|         M4|         17|    GG1919|     Finland|
|         5|         M5|          3|   ACMAX22|   Hong Kong|
|         6|         M6|          9|    AC1000|   Singapore|
|         7|         M7|         13|    FN39TG|South Africa|
|         8|         M8|         25|    JDNS77|   Australia|
|         9|         M9|         11|    GG1919|      Mexico|
|        10|        M10|         23|   ACMAX22|       China|
|        11|        M11|         14|    AC1000|     Belgium|
|        12|        M12|         26|    FN39TG|     Finland|
|        13|        M13|         25|    JDNS77|Saudi Arabia|
|        14|        M14|         17|    GG1919|     Germany|
|        15|        M15|         19|   ACMAX22|      Israel|
|        16|        M16|         23|    AC1000|      Turkey|
|        17|        M17|         11|    FN39TG|       Egypt|
|        18|        M18|         25|    JDNS77|   Indonesia|
|        19|        M19|         14|    GG1919|      Canada|
|        20|        M20|         19|   ACMAX22|   Argentina|
+----------+-----------+-----------+----------+------------+

buildings table registered!
18/05/28 16:06:56 INFO FileSourceStrategy: Pruning directories with:
```

Task 3.1 Figure out the number of times, temperature has changed by 5 degrees or more for each country

Solution Approach –

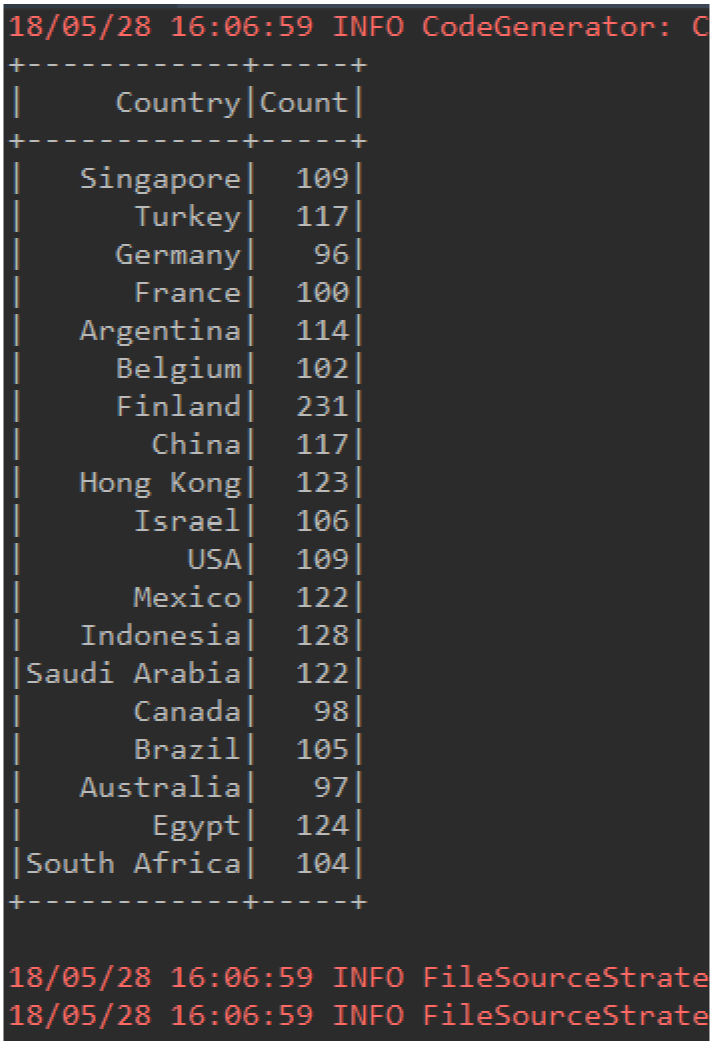
- We need to join both the tables on key building ID
- We have already created a column 'tempchange' in task 1.2 , select tempchange where it is = 1 and group on country to get the count

Approach 1 : Using SQL Queries



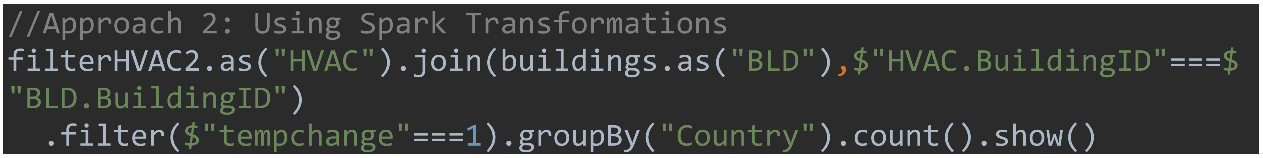
```
//Approach 1: Using SQL Queries
filterHVAC2.registerTempTable("FilteredHVAC")
spark.sql("Select Country,Count(Country) as Count from FilteredHVAC
HVAC " +
  "Join building_table BLD On HVAC.BuildingID=BLD.BuildingID where
tempchange=1 group by country").show()
```

Output



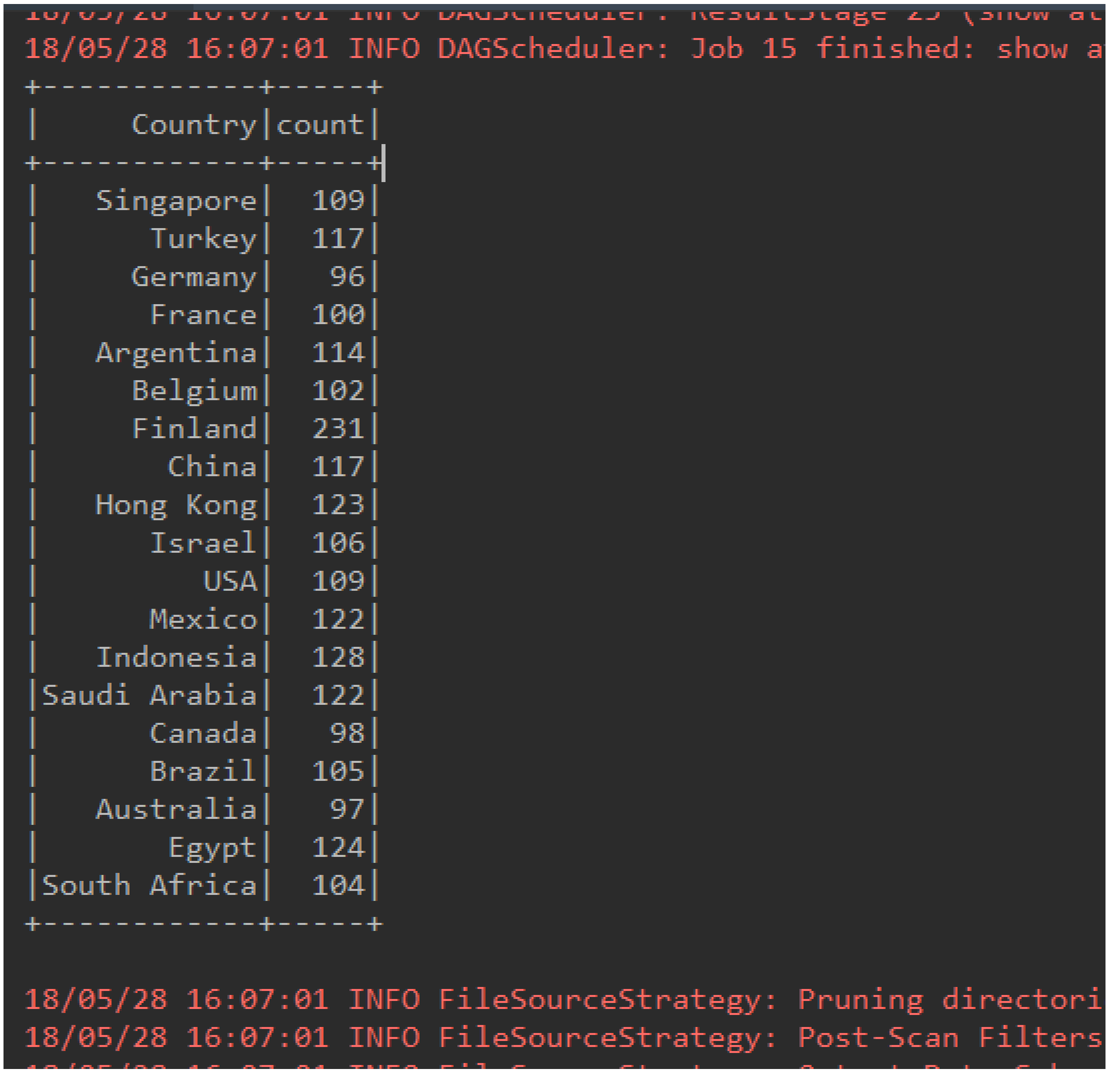
```
18/05/28 16:06:59 INFO CodeGenerator: C
+------------+-----+
|     Country|Count|
+------------+-----+
|   Singapore|  109|
|      Turkey|  117|
|     Germany|   96|
|      France|  100|
|   Argentina|  114|
|     Belgium|  102|
|     Finland|  231|
|       China|  117|
|   Hong Kong|  123|
|      Israel|  106|
|         USA|  109|
|      Mexico|  122|
|   Indonesia|  128|
|Saudi Arabia|  122|
|      Canada|   98|
|      Brazil|  105|
|   Australia|   97|
|       Egypt|  124|
|South Africa|  104|
+------------+-----+

18/05/28 16:06:59 INFO FileSourceStrate
18/05/28 16:06:59 INFO FileSourceStrate
```

Approach 2: Using Spark SQL Transformations → join , filter , groupby , count



```
//Approach 2: Using Spark Transformations
filterHVAC2.as("HVAC").join(buildings.as("BLD"),$"HVAC.BuildingID"===$
"BLD.BuildingID")
  .filter($"tempchange"===1).groupBy("Country").count().show()
```

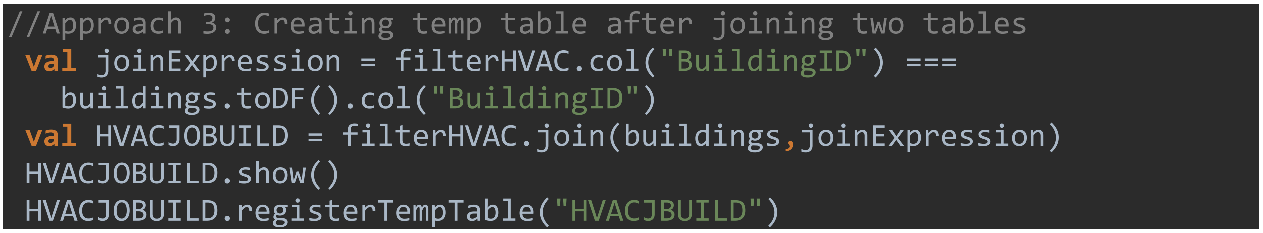
Output



```
18/05/28 16:07:01 INFO DAGScheduler: ResultStage 25 (show at
18/05/28 16:07:01 INFO DAGScheduler: Job 15 finished: show a
+------------+-----+
|     Country|count|
+------------+-----+
|   Singapore|  109|
|      Turkey|  117|
|     Germany|   96|
|      France|  100|
|   Argentina|  114|
|     Belgium|  102|
|     Finland|  231|
|       China|  117|
|   Hong Kong|  123|
|      Israel|  106|
|         USA|  109|
|      Mexico|  122|
|   Indonesia|  128|
|Saudi Arabia|  122|
|      Canada|   98|
|      Brazil|  105|
|   Australia|   97|
|       Egypt|  124|
|South Africa|  104|
+------------+-----+

18/05/28 16:07:01 INFO FileSourceStrategy: Pruning directori
18/05/28 16:07:01 INFO FileSourceStrategy: Post-Scan Filters
18/05/28 16:07:01 INFO FileSourceStrategy: Output Data Sch
```

Approach 3:

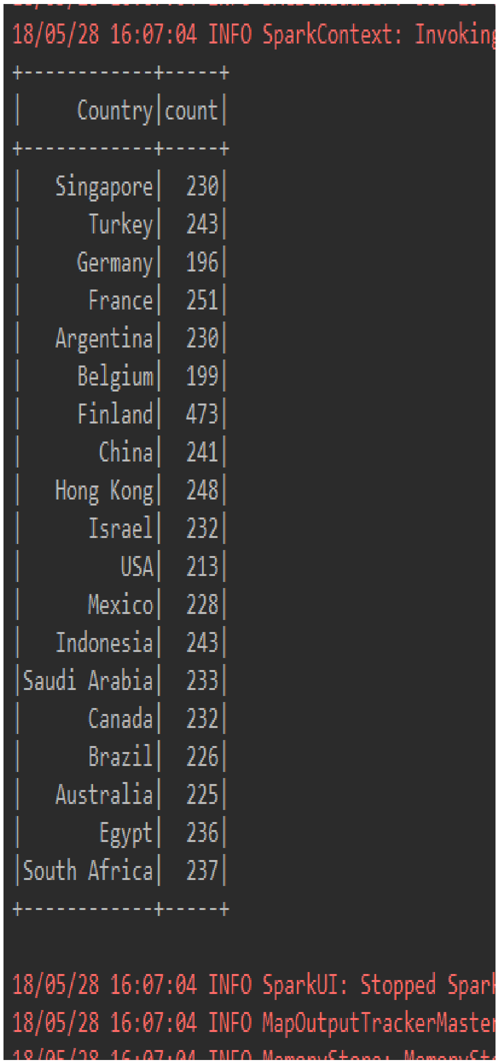
We can also create a Dataframe by joining two tables and register the newly created table joining and perform the quering on the same.



```
//Approach 3: Creating temp table after joining two tables
 val joinExpression = filterHVAC.col("BuildingID") ===
   buildings.toDF().col("BuildingID")
 val HVACJOBUILD = filterHVAC.join(buildings,joinExpression)
 HVACJOBUILD.show()
 HVACJOBUILD.registerTempTable("HVACJBUILD")
 val selective = spark.sql("""select tempchange, Country from
HVACJBUILD WHERE
                             tempchange = 1""").toDF()

 // registering the nwly created table
 selective.registerTempTable("newselective")

//Approach 3.1
 spark.sql("""select Country, count(tempchange) from newselective
group by
 Country""").show()

 //Approach 3.2
 selective.filter($"tempchange"===1).groupBy("Country").count().show()
```

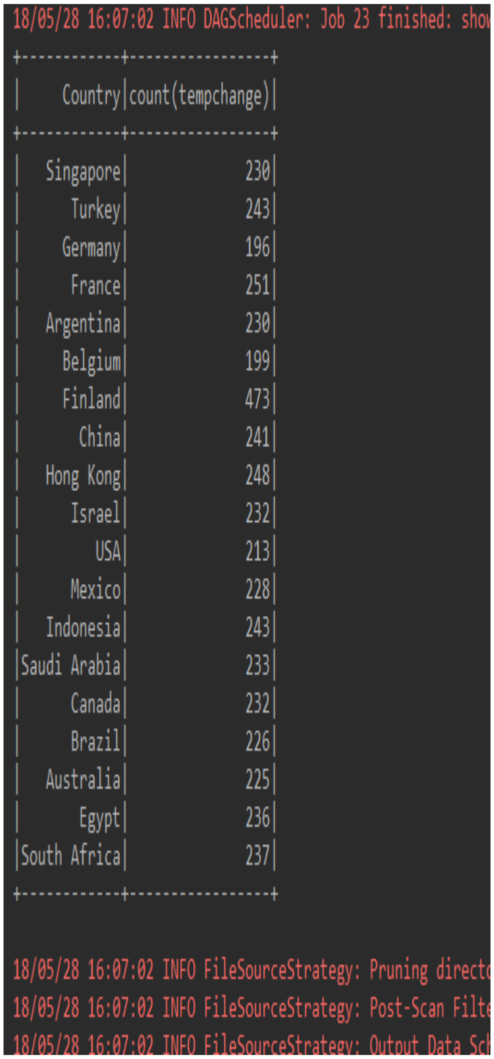
Output

Approach 3.2



```
18/05/28 16:07:04 INFO SparkContext: Invoking
+------------+-----+
|     Country|count|
+------------+-----+
|   Singapore|  230|
|      Turkey|  243|
|     Germany|  196|
|      France|  251|
|   Argentina|  230|
|     Belgium|  199|
|     Finland|  473|
|       China|  241|
|   Hong Kong|  248|
|      Israel|  232|
|         USA|  213|
|      Mexico|  228|
|   Indonesia|  243|
|Saudi Arabia|  233|
|      Canada|  232|
|      Brazil|  226|
|   Australia|  225|
|       Egypt|  236|
|South Africa|  237|
+------------+-----+

18/05/28 16:07:04 INFO SparkUI: Stopped Spark
18/05/28 16:07:04 INFO MapOutputTrackerMaster
18/05/28 16:07:04 INFO MemoryStore: MemorySt
```

Approach 3.1



```
18/05/28 16:07:02 INFO DAGScheduler: Job 23 finished: show
+------------+-----------------+
|     Country|count(tempchange)|
+------------+-----------------+
|   Singapore|              230|
|      Turkey|              243|
|     Germany|              196|
|      France|              251|
|   Argentina|              230|
|     Belgium|              199|
|     Finland|              473|
|       China|              241|
|   Hong Kong|              248|
|      Israel|              232|
|         USA|              213|
|      Mexico|              228|
|   Indonesia|              243|
|Saudi Arabia|              233|
|      Canada|              232|
|      Brazil|              226|
|   Australia|              225|
|       Egypt|              236|
|South Africa|              237|
+------------+-----------------+

18/05/28 16:07:02 INFO FileSourceStrategy: Pruning directo
18/05/28 16:07:02 INFO FileSourceStrategy: Post-Scan Filte
18/05/28 16:07:02 INFO FileSourceStrategy: Output Data Sch
```