Adapt MV datamap to Spark 2.1 version

The Class we cann’t access in Spark 2.1 version

1. org.apache.spark.internal.Logging
2. org.apache.spark.sql.internal.SQLConf

Solution:Create class extends above classed in spark2.1 modular

```
package org.apache.spark

trait Logging extends org.apache.spark.internal.Logging{
}
```

```
package org.apache.spark.sql

class SQLConf extends org.apache.spark.sql.internal.SQLConf {
}
```

The Class that Spark 2.1 version doesn’t have

1. org.apache.spark.sql.catalyst.plans.logical.Subquery
2. org.apache.spark.sql.catalyst.catalog.interface.HiveTableRelation

Solution:Create this class in spark2.1 modular

```
   package org.apache.spark.sql.catalyst.plans.logical

   import org.apache.spark.sql.catalyst.expressions.Attribute

   /**
     * This node is inserted at the top of a subquery when it is optimized.
This makes sure we can
     * recognize a subquery as such, and it allows us to write subquery aware
transformations.
     */
   case class Subquery(child: LogicalPlan) extends UnaryNode {
     override def output: Seq[Attribute] = child.output
   }
```

Solution: Use CatalogRelation instead and don’t use (in LogicalPlanSignatureGenerator)

The method that we can’t access in Spark 2.1 version

1. sparkSession.sessionState.catalog.lookupRelation

Solution:Add this method to CarbonToSparkAdapter in spark2.1 modular

```
 def lookupRelation(sparkSession: SparkSession, tableIdentifier:
TableIdentifier): LogicalPlan = {
    sparkSession.sessionState.catalog.lookupRelation(tableIdentifier)
  }
```

The changes of some class

1. org.apache.spark.sql.catalyst.expressions.SortOrder
2. org.apache.spark.sql.catalyst.expressions.Cast
3. org.apache.spark.sql.catalyst.plans.Statistics

Solution: Adapt the new interface

The method that Spark 2.1 version doesn’t have

1. normalizeExprId,canonicalized of org.apache.spark.sql.catalyst.plans.QueryPlan
2. CASE_SENSITIVE of SQLConf
3. STARSCHEMA_DETECTION of SQLConf

Solution:Don’t use normalize , canonicalize and the CASE_SENSITIVE, STARSCHEMA_DETECTION

Some logicplan optimization rules that Spark 2.1 version doesn’t have

1. SimplifyCreateMapOps
2. SimplifyCreateArrayOps
3. SimplifyCreateStructOps
4. RemoveRedundantProject
5. RemoveRedundantAliases
6. PullupCorrelatedPredicates
7. ReplaceDeduplicateWithAggregate
8. EliminateView

Solution: Delete or move the code to carbon project

Generate the instance in SparkSQLUtil to adapt Spark 2.1 version

```
  def getReorderJoinObj(conf: SQLConf): Rule[LogicalPlan] = {
    if (SparkUtil.isSparkVersionEqualTo("2.2")) {
      val className = "org.apache.spark.sql.catalyst.optimizer.ReorderJoin";
      CarbonReflectionUtils.createObject(className,
conf)._1.asInstanceOf[Rule[LogicalPlan]]
    } else if (SparkUtil.isSparkVersionXandAbove("2.3")) {
      val className = "org.apache.spark.sql.catalyst.optimizer.ReorderJoin$";
      CarbonReflectionUtils.createObjectOfPrivateConstructor(className)._1
        .asInstanceOf[Rule[LogicalPlan]]
    } else if (SparkUtil.isSparkVersionEqualTo("2.1")) {
      val className = "org.apache.spark.sql.catalyst.optimizer.ReorderJoin$";
      CarbonReflectionUtils.createObjectOfPrivateConstructor(className)._1
        .asInstanceOf[Rule[LogicalPlan]]
    }
    else {
      throw new UnsupportedOperationException("Spark version not supported")
    }
  }
```

```
  def getEliminateOuterJoinObj(conf: SQLConf): Rule[LogicalPlan] = {
    if (SparkUtil.isSparkVersionEqualTo("2.2")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.EliminateOuterJoin";
      CarbonReflectionUtils.createObject(className,
conf)._1.asInstanceOf[Rule[LogicalPlan]]
    } else if (SparkUtil.isSparkVersionXandAbove("2.3")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.EliminateOuterJoin$";
      CarbonReflectionUtils.createObjectOfPrivateConstructor(className)._1
        .asInstanceOf[Rule[LogicalPlan]]
    } else if (SparkUtil.isSparkVersionEqualTo("2.1")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.EliminateOuterJoin$";
      CarbonReflectionUtils.createObjectOfPrivateConstructor(className)._1
        .asInstanceOf[Rule[LogicalPlan]]
    }
    else {
      throw new UnsupportedOperationException("Spark version not supported")
    }
  }
```

```
  def getNullPropagationObj(conf: SQLConf): Rule[LogicalPlan] = {
    if (SparkUtil.isSparkVersionEqualTo("2.2")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.NullPropagation";
      CarbonReflectionUtils.createObject(className,
conf)._1.asInstanceOf[Rule[LogicalPlan]]
    } else if (SparkUtil.isSparkVersionXandAbove("2.3")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.NullPropagation$";
      CarbonReflectionUtils.createObjectOfPrivateConstructor(className)._1
        .asInstanceOf[Rule[LogicalPlan]]
    } else if (SparkUtil.isSparkVersionEqualTo("2.1")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.NullPropagation$";
      CarbonReflectionUtils.createObjectOfPrivateConstructor(className)._1
        .asInstanceOf[Rule[LogicalPlan]]
    } else {
      throw new UnsupportedOperationException("Spark version not supported")
    }
  }
```

```
  def getCheckCartesianProductsObj(conf: SQLConf): Rule[LogicalPlan] = {
    if (SparkUtil.isSparkVersionEqualTo("2.2")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.CheckCartesianProducts";
      CarbonReflectionUtils.createObject(className,
conf)._1.asInstanceOf[Rule[LogicalPlan]]
    } else if (SparkUtil.isSparkVersionXandAbove("2.3")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.CheckCartesianProducts$";
      CarbonReflectionUtils.createObjectOfPrivateConstructor(className)._1
        .asInstanceOf[Rule[LogicalPlan]]
    } else if (SparkUtil.isSparkVersionEqualTo("2.1")) {
      val className =
"org.apache.spark.sql.catalyst.optimizer.CheckCartesianProducts";
      CarbonReflectionUtils.createObject(className,
conf)._1.asInstanceOf[Rule[LogicalPlan]]
    }
    else {
      throw new UnsupportedOperationException("Spark version not supported")
    }
  }
```

Query SQL pass the MV check in Spark 2.1 version(CarbonSessionState)

```
class CarbonAnalyzer(catalog: SessionCatalog,
    conf: CatalystConf,
    sparkSession: SparkSession,
    analyzer: Analyzer) extends Analyzer(catalog, conf) {

  val mvPlan = try {
    CarbonReflectionUtils.createObject(
      "org.apache.carbondata.mv.datamap.MVAnalyzerRule",
      sparkSession)._1.asInstanceOf[Rule[LogicalPlan]]
  } catch {
    case e: Exception =>
      null
  }

  override def execute(plan: LogicalPlan): LogicalPlan = {
    var logicalPlan = analyzer.execute(plan)
    logicalPlan =
CarbonPreAggregateDataLoadingRules(sparkSession).apply(logicalPlan)
    CarbonPreAggregateQueryRules(sparkSession).apply(logicalPlan)

    if (mvPlan != null) {
      mvPlan.apply(logicalPlan)
    } else {
      logicalPlan
    }
  }
}
```