Spark SQL

OutLine

Spark SQL理论

Spark SQL实践

Spark SQL基础简介

- Spark SQL是Apache Spark大数据框架一部分，主要用于处理结构化数据和对spark数据执行类sql的查询
- Spark为其提供了一个称为DataFrame的编程抽象，充当分布式SQL查询引擎

- Spark SQL功能：
  - 集成：无缝将SQL查询与Spark程序混合
  - 统一数据访问：加载来自各种来源的数据
  - 兼容性：Spark SQL重用Hive前端和MetaStore，与现有Hive数据、查询和UDF的安全兼容，只需要和Hive一期安装即可
  - 标准连接：JDBC和ODBC
  - 扩展性：对交互式查询和长查询使用相同的引擎，Spark SQL利用RDD模型来支持查询容错，使其扩展大大型作业

Spark SQL框架

- Spark SQL框架：
  - 把数据读入到SparkSQL中，SparkSQL进行数据处理或算法实现，然后再把处理后的数据输出到相应的输出源

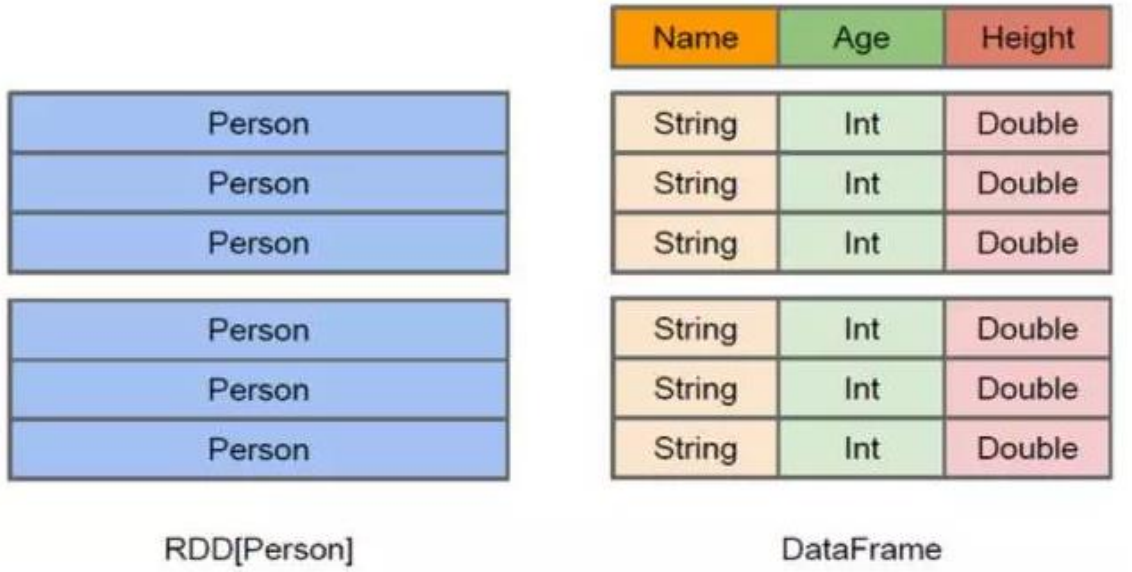
  - Input：
    - 数据源丰富：Hive、json、txt、JDBC等
    - Spark SQL存在两个类进行对接：HiveContext和SQLContext，其中HiveContext继承了SQLContext的所有方法，同时又进行扩展
    - SQLContext用于对接绝大多类型数据源， HiveContext是SQLContext的超集
    - Spark SQL处理读入的数据，采用的是DataFrame中提供的方法

DataFrame

- 作为2014–2015年Spark最大的API改动，DataFrame能够使得大数据更为简单
- 之前Spark SQL API的SchemaRDD已经更名为DataFrame
- 分布式的数据集合，按照命名列的形式组织数据
- 通过调用将DataFrame的内容作为行RDD（RDD of Rows）返回的rdd方法，可以将DataFrame转换成RDD
- 通过如下方式创建DataFrame：
  - 已有RDD
  - 结构化数据文件
  - Json数据
  - Hive表
  - 外部数据库

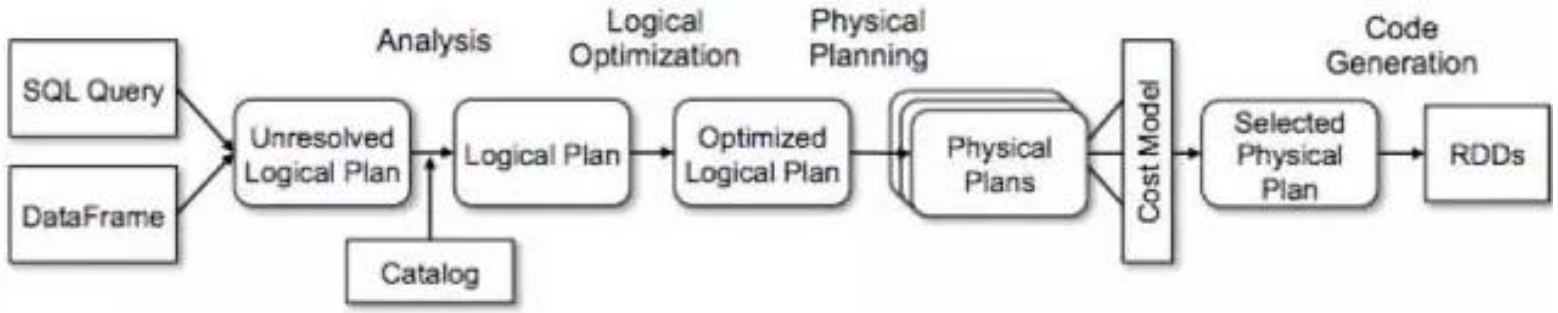
DataFrame

- RDD以record为单位，spark优化时无法洞悉record内部的细节，无法深度优化，限制sparkSQL性能的提升；DataFrame包含了每个record的metadata元数据信息，DataFrame的优化可以对列内部优化
- DataFrame是一个以命名列方式组织的分布式数据集，等同于关系型数据库中的一个表



DataFrame

- DataFrame是基于RDD的抽象
- DataFrame的底层结构是RDD. Spark在你使用DataFrame时会优化你的代码



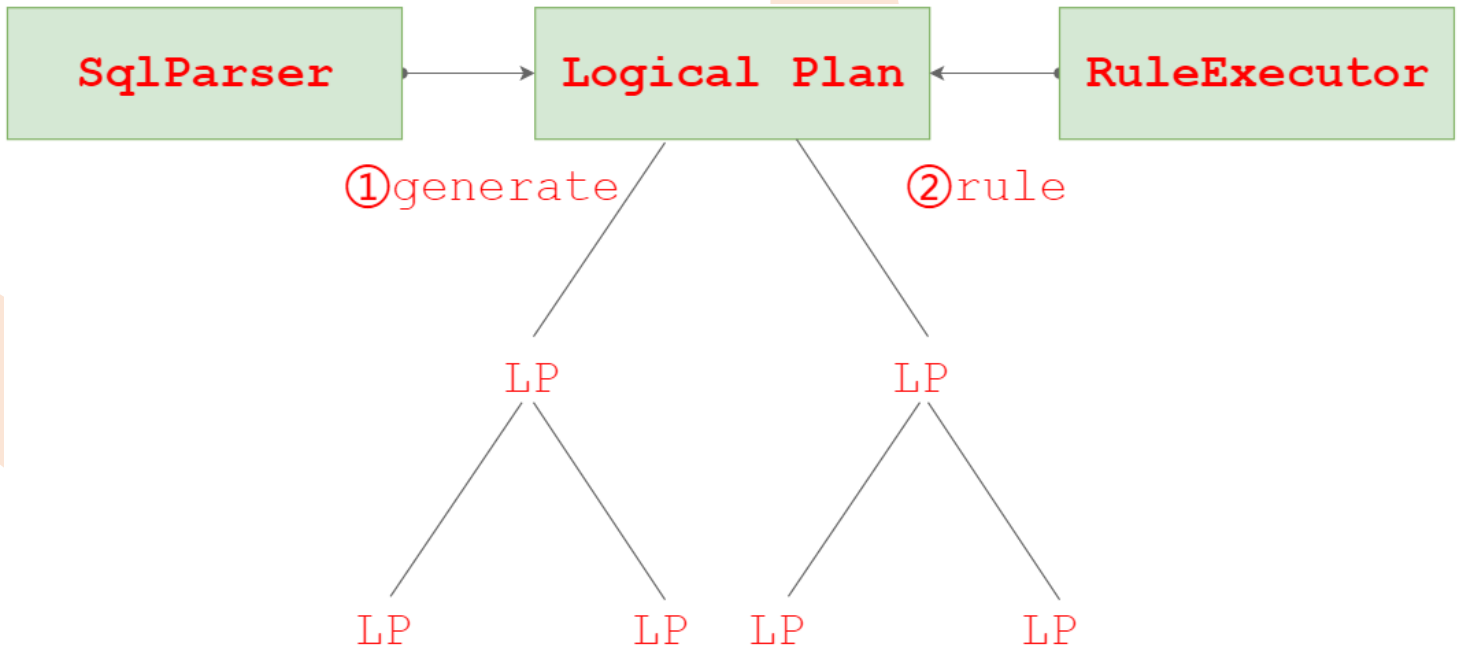
- Spark对于DataFrame在执行时间和内存使用上对于RDD有极大的优化
  - Catalyst优化引擎：使得执行时间减少75%
  - Project Tungsten Off-heap内存管理：是内存使用量减少75%，无垃圾回收器
- 使用python及scala执行RDD的速度明显比DataFrame慢
- 但同样对于DataFrame，两种语言没有区别，两者性能均优于普通Python RDD实现的4倍，也达到了Scala RDD实现的两倍

优化引擎

- Catalyst
- SQL优化器核心执行策略两个方向：规则和代价
- 基于规则：经验式、启发式地优化思路，更多地依靠前辈总结出来的优化规则，简单易行且能够覆盖到大部分优化逻辑
- 基于代价：核心算子优化

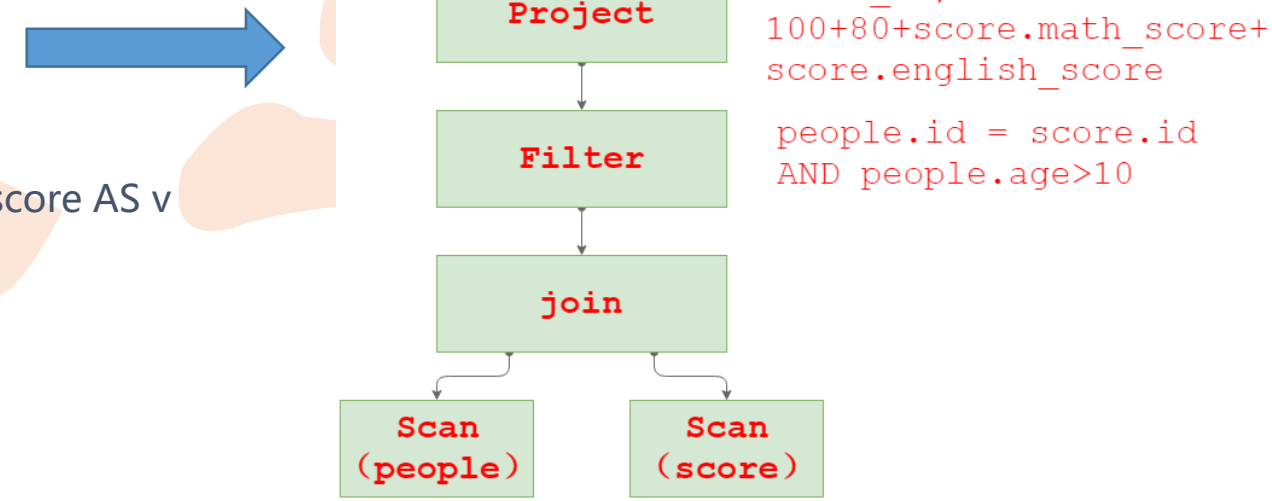
  两个表执行Join应该使用

  BroadcastHashJoin还是SortMergeJoin?



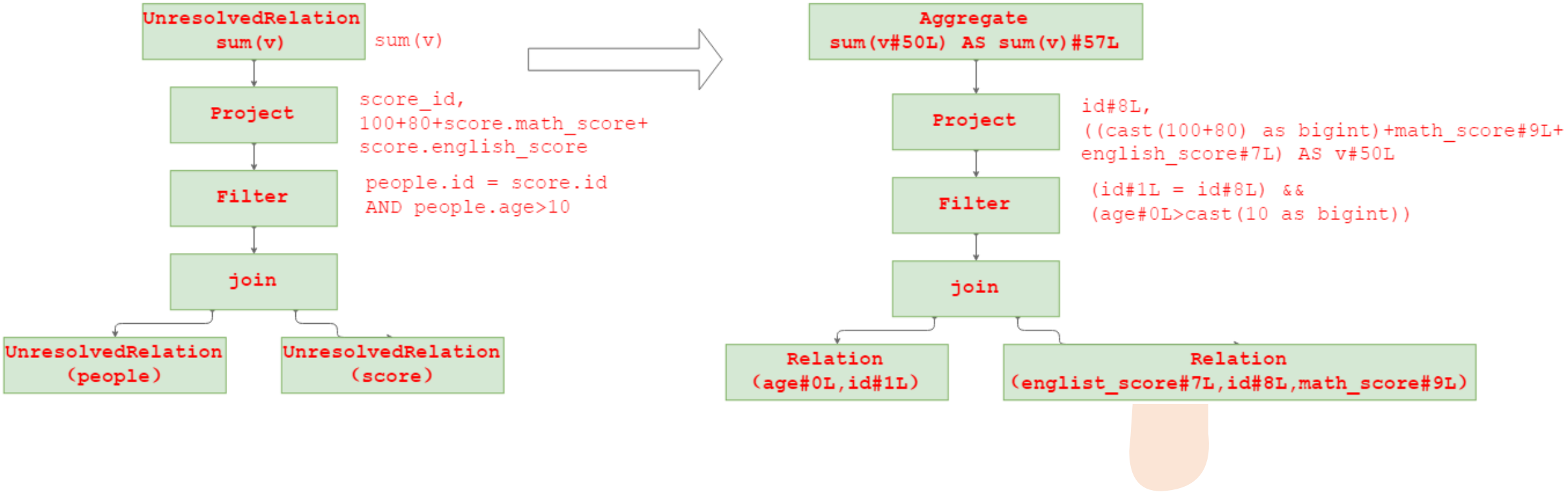
实现分析

- Parser（解析器）：SqlParser生成LogicPlan Tree
  - 主要先进行词法分析，再进行语法分析
    - 词法分析：讲输入的sql语句串解析为一个一个的token
    - 语法分析：再词法分析基础上，将单词序列组合成各类语法短语，组成各个LogicPlan
  - 例子：
    - SELECT sum(v)
    - FROM(
    - SELECT score.id,
    - 100+80+score.math_score+score.english_score AS v
    - FROM people JOIN score
    - WHERE people.id=score.id
    - AND people.age>10
    -) a



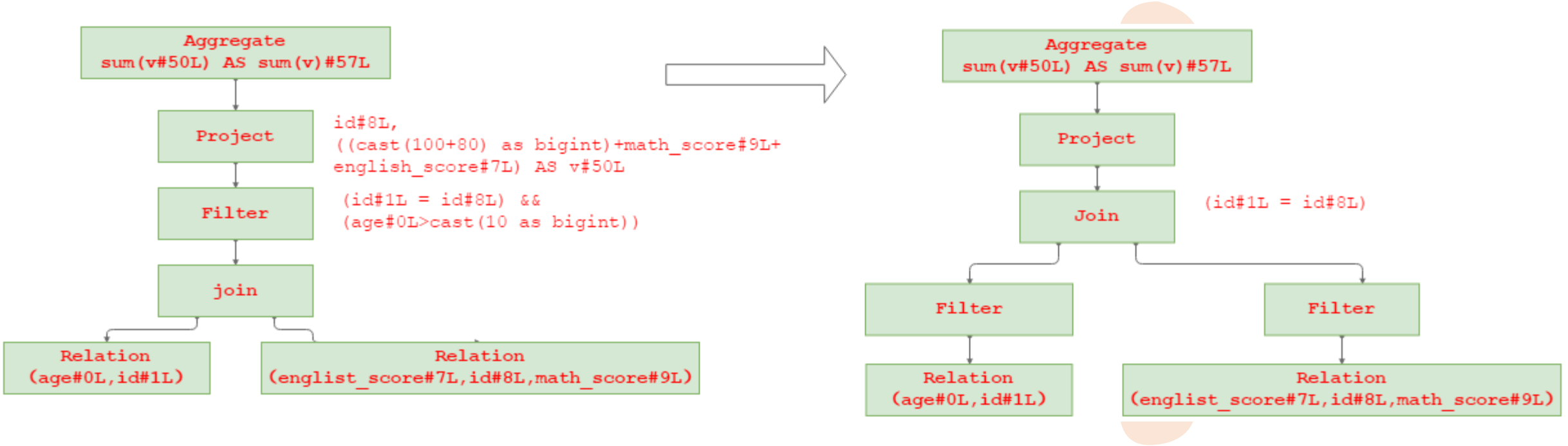
实现分析

- Analyzer：遍历整个语法树，对树上的每个节点进行数据类型绑定以及函数绑定
  - 根据元数据表解析为包含必要列的表，并且相应字段解析为相应的数据类型，相应的计算逻辑解析为对应的函数



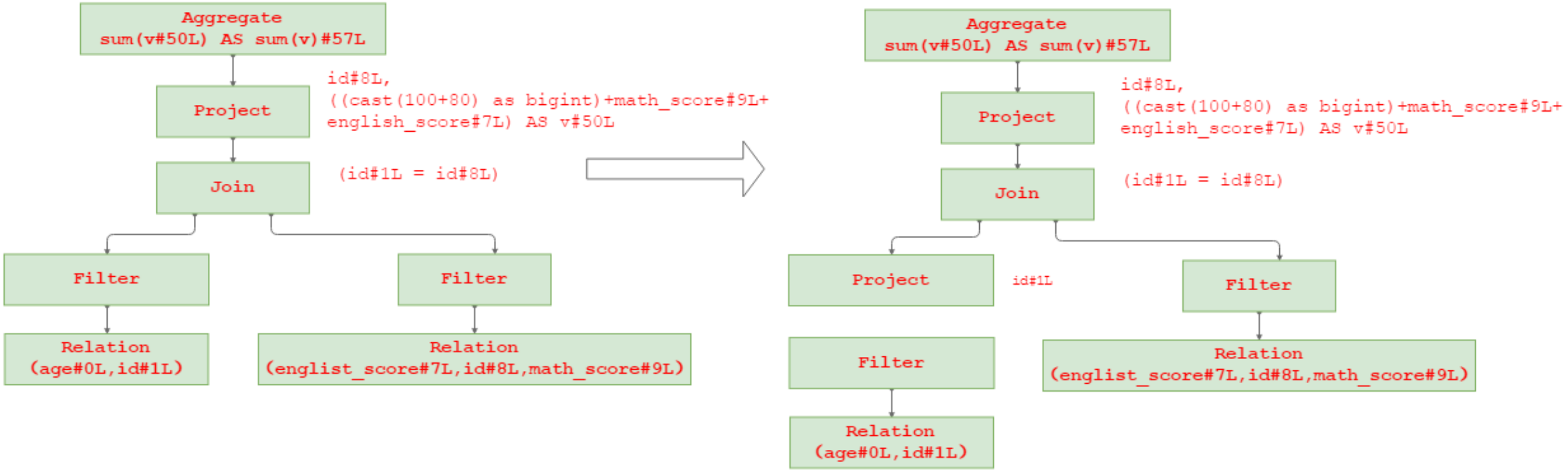
实现分析

- Optimizer：是Catalyst的核心
  - 基于规则优化实际上对语法树再做一次遍历，模式匹配能够满足特定细节的节点，再进行相应的等价变变换
  - 经典规则：谓词下推（Predicate Pushdown）、常量累加（Constant Folding）和列值裁剪（Column Pruning）



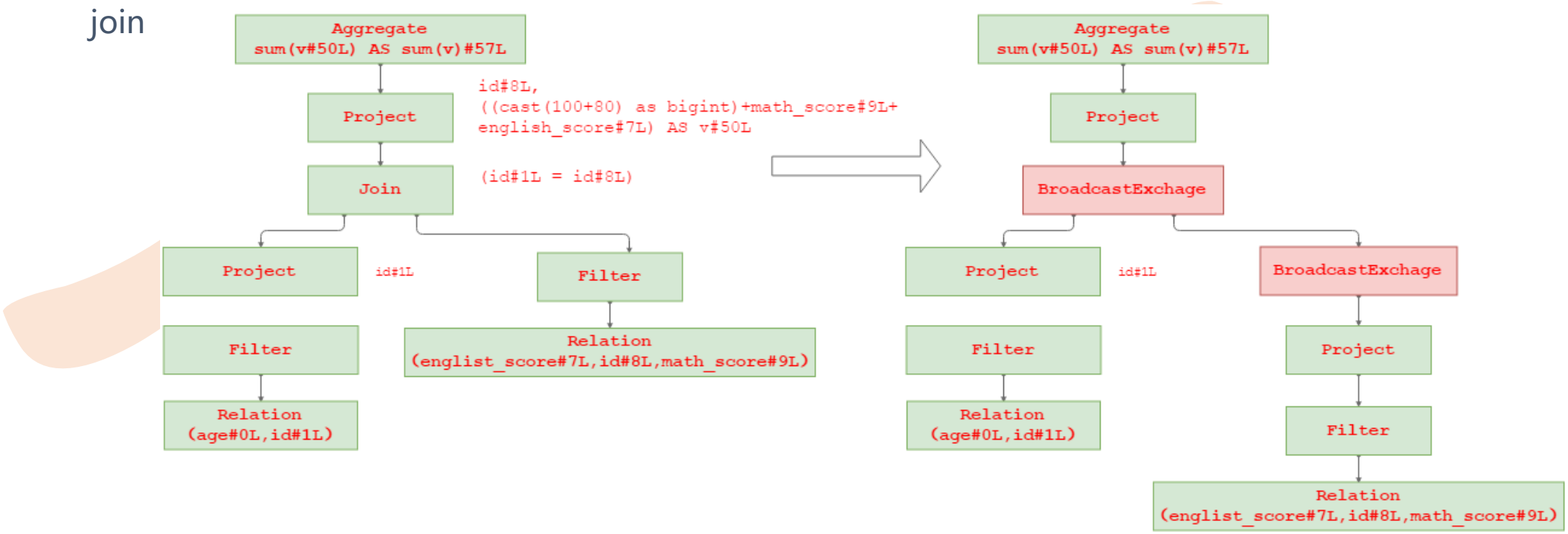
实现分析

- Optimizer：是Catalyst的核心
  - 基于规则优化实际上对语法树再做一次遍历，模式匹配能够满足特定细节的节点，再进行相应的等价变变换
  - 经典规则：谓词下推（Predicate Pushdown）、常亮累加（Constant Folding）和列值裁剪（Column Pruning）



实现分析

- Physical Planning：物理计划层
  - 用物理操作算子产生一个或者多个物理计划。然后用cost模型选择一个物理计划。目前基于cost-based的优化仅仅用于选择join算法：对已知的很小的relations，sparksql会选择使用spark的提供的点对点的广播功能实现Broadcast join



实现分析

- Spark SQL执行计划：
- 使用queryExecution方法查看


```
scala> spark.sql("select p.name,s.math_score from people p , score s where p.id = s.id and p.id > 1").queryExecution
res2: org.apache.spark.sql.execution.QueryExecution =
== Parsed Logical Plan ==
'Project ['p.name, 's.math_score]
+- 'Filter (('p.id = 's.id) && ('p.id > 1))
   +- 'Join Inner
      :- 'UnresolvedRelation `people`, p
      +- 'UnresolvedRelation `score`, s

== Analyzed Logical Plan ==
Project [name#2, math_score#10L]
+- Filter ((id#1L = id#9L) && (id#1L > cast(1 as bigint)))
   +- Join Inner
      :- SubqueryAlias p
      :  +- SubqueryAlias people
      :     +- Relation[age#0L,id#1L,name#2] json
      +- SubqueryAlias s
         +- SubqueryAlias score
            +- Relation[english_score#8L,id#9L,math_score#10L] json

== Optimized Logical Plan ==
Project [name#2, math_score#10L]
+- Join Inner, (id#1L = id#9L)
   :- Project [id#1L, name#2]
   :  +- Fil...
scala> spark.sql("select p.name,s.math_score from people p , score s where p.id = s.id and p.id > 1").explain
== Physical Plan ==
*Project [name#2, math_score#10L]
+- *BroadcastHashJoin [id#1L], [id#9L], Inner, BuildRight
   :- *Project [id#1L, name#2]
   :  +- *Filter (isnotnull(id#1L) && (id#1L > 1))
   :     +- *FileScan json [id#1L,name#2] Batched: false, Format: JSON, Location: InMemoryFileIndex[file:/Users/libisthanks/Docu
, PushedFilters: [IsNotNull(id), GreaterThan(id,1)], ReadSchema: struct<id:bigint,name:string>
   +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, true]))
      +- *Project [id#9L, math_score#10L]
```

Q & A

@八斗学院