Spark SQL 架构体系与执行流程

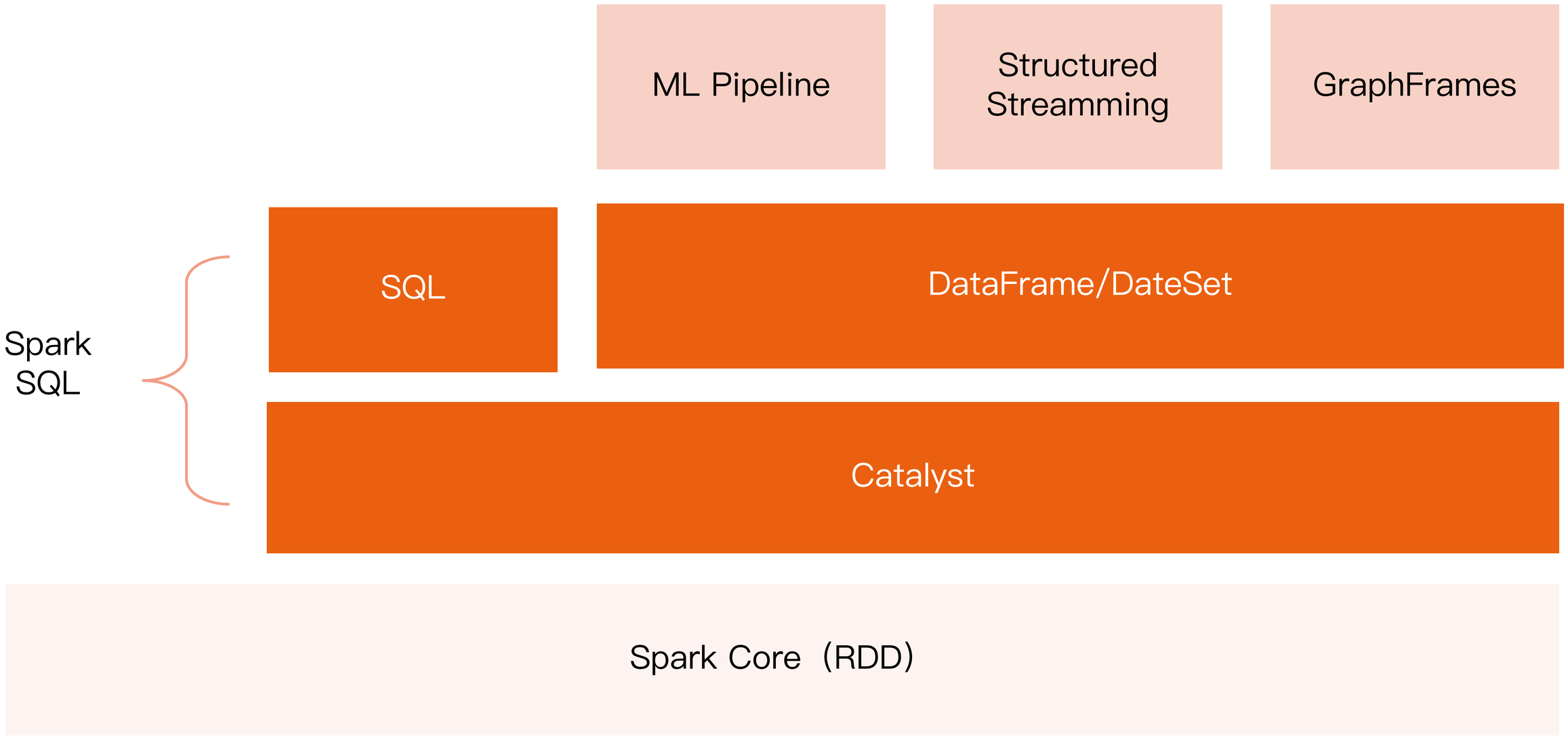
极客时间

金澜涛

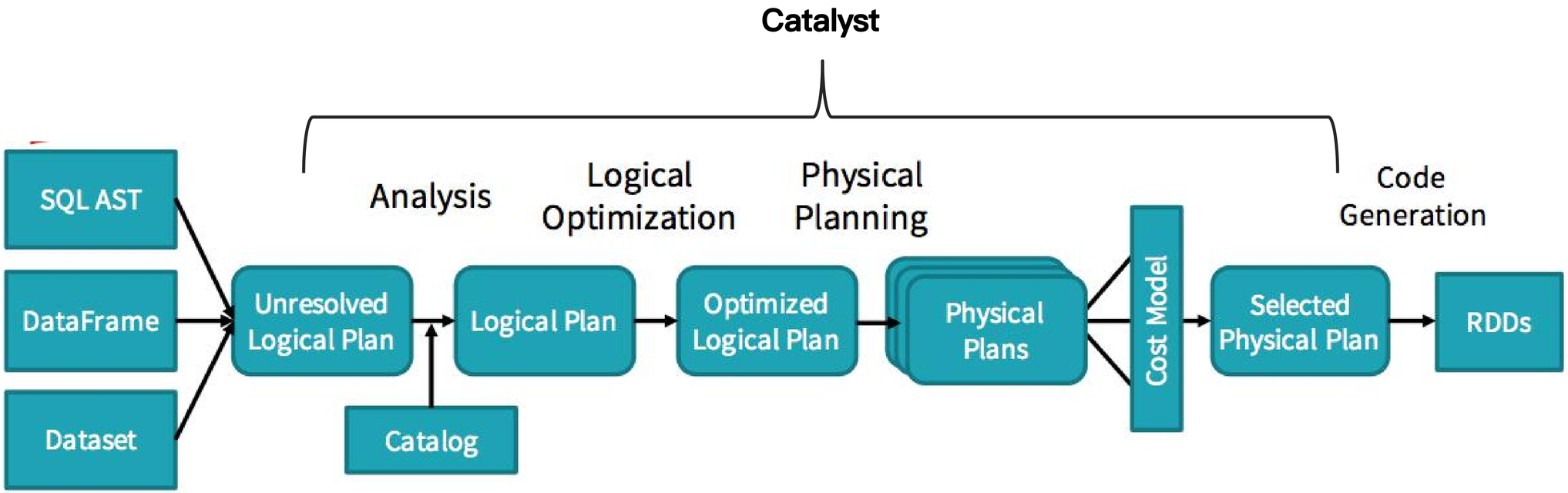
目录

Spark SQL 的基本架构

Spark SQL 架构体系



Catalyst


Catalyst
SQL AST
DataFrame
Dataset
Analysis
Unresolved Logical Plan
Catalog
Logical Optimization
Logical Plan
Physical Planning
Optimized Logical Plan
Physical Plans
Cost Model
Code Generation
Selected Physical Plan
RDDs

SQL 编译器和 ANTLR

为什么要使用 DSL

DSL 是领域专用语言，常见的 DSL 有 SQL、CSS、Shell 等等，这些 DSL 语言有别于其他通用语言如 C++、Java、C# 等，DSL 常在特殊的场景或领域中使用。

领域专用语言通常是被领域专家使用，领域专家一般不熟悉通用编程语言，但是他们一般对业务非常了解。

DSL	领域专家
通用编程语言	程序员
机器语言	电脑

ANTLR

ANTLR（Another Tool for Language Recognition）是可以根据输入自动生成语法树并可视化的显示出来的开源语法分析器。

ANTLR 主要包含词法分析器，语法分析器和树分析器。

词法分析器又称 Scanner、Lexer 或 Tokenizer。词法分析器的工作是分析量化那些本来毫无意义的字符流，将他们翻译成离散的字符组（也就是一个一个的Token），包括关键字、标识符、符号（symbols）和操作符供语法分析器使用。

语法分析器又称编译器。在分析字符流的时候，Lexer 不关心所生成的单个 Token 的语法意义及其与上下文之间的关系。语法分析器则将收到的 Token 组织起来，并转换成为目标语言语法定义所允许的序列。

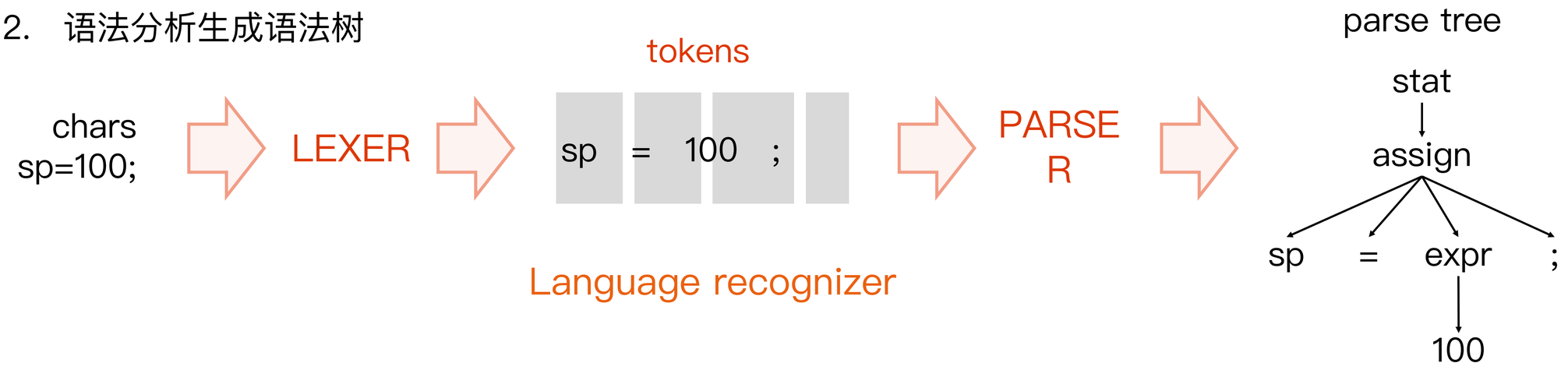
树分析器可以用于对语法分析生成的抽象语法树进行遍历，并能执行一些相关的操作。

Spark Parser

Spark Parser 简单来说是利用 ANTLR4 将 SQL 字符串切分成一个一个的 Token，再根据一定语义规则将其解析为一棵语法树 AST。

下图是一个示例性的 SQL 语句（有两张表，其中 people 表主要存储用户基本信息， score 表存储用户的各种成绩），通过 Parser 解析后的 AST 语法树。

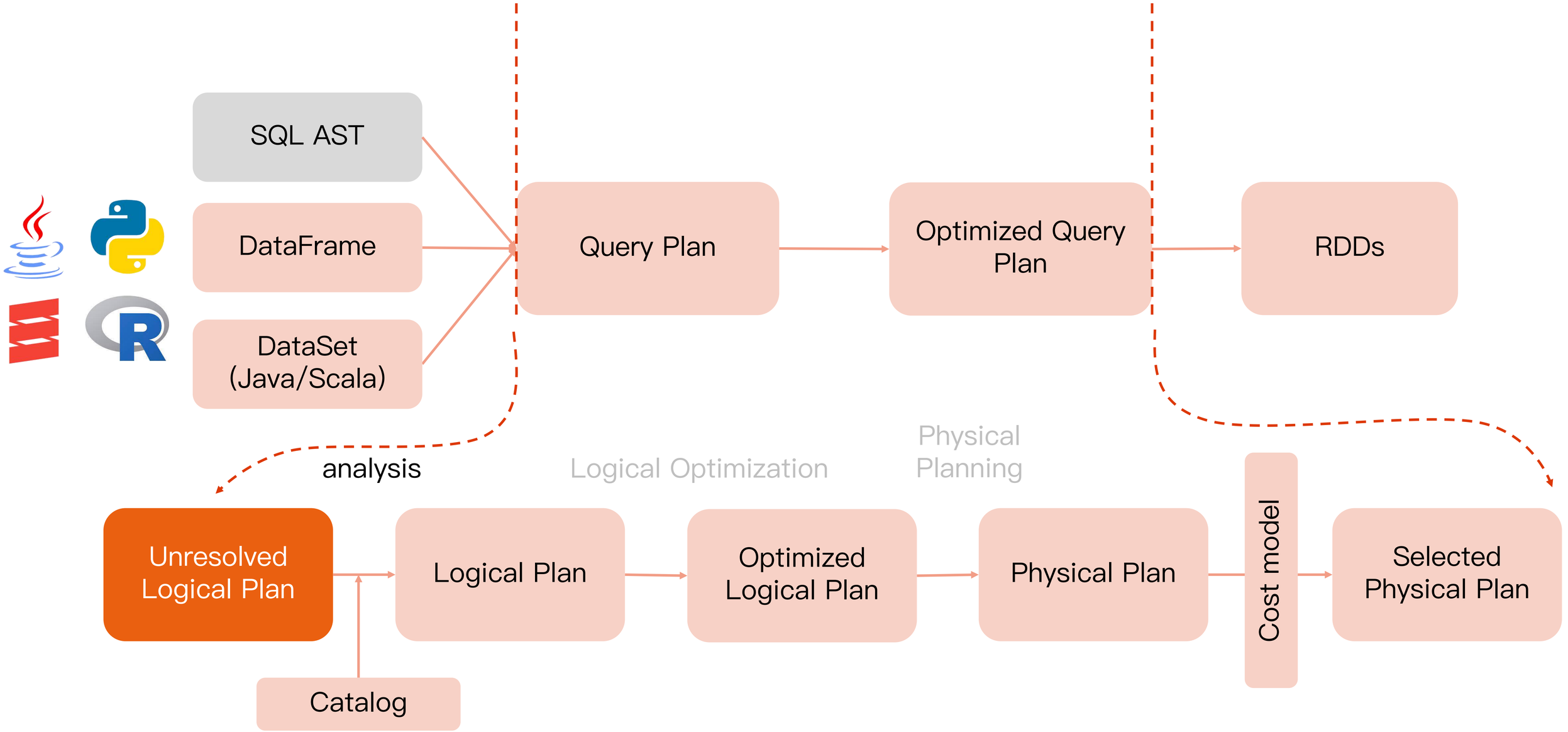
1. 词法分析生成 Token 流
2. 语法分析生成语法树



Spark Parser

1. sql text 经过 SqlParser 解析成 Unresolved LogicalPlan.

Spark Parser



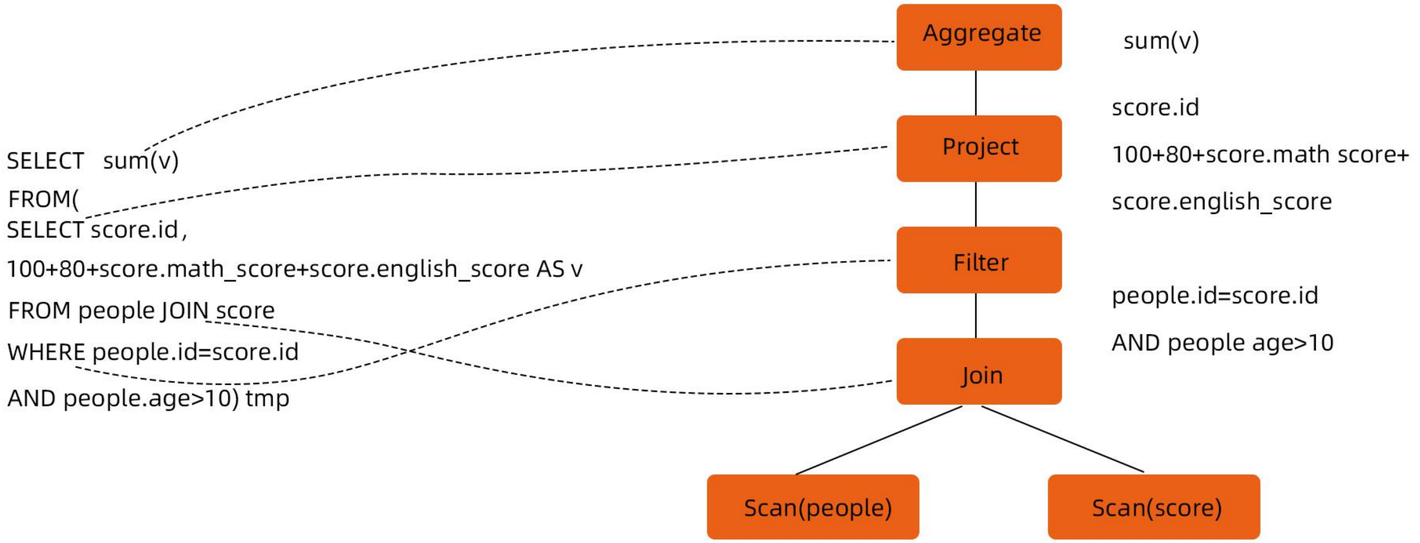
Unresolved Logical Plan

生成语法树之后，使用 AstBuilder 将语法树转换成 LogicalPlan，这个 LogicalPlan 也被称为 Unresolved LogicalPlan。解析后的逻辑计划如下：

```
== Parsed Logical Plan ==
'Project [unresolvedalias('sum('v), None)]
+- 'SubqueryAlias `iteblog`
   +- 'Project ['t1.id, ((1 + 2) + 't1.value) AS v#16]
      +- 'Filter ((('t1.id = 't2.id) && ('t1.cid = 1)) && (('t1.did = ('t1.cid + 1)) &&
('t2.id > 5)))
         +- 'Join Inner
            :- 'UnresolvedRelation `t1`
            +- 'UnresolvedRelation `t2`
```

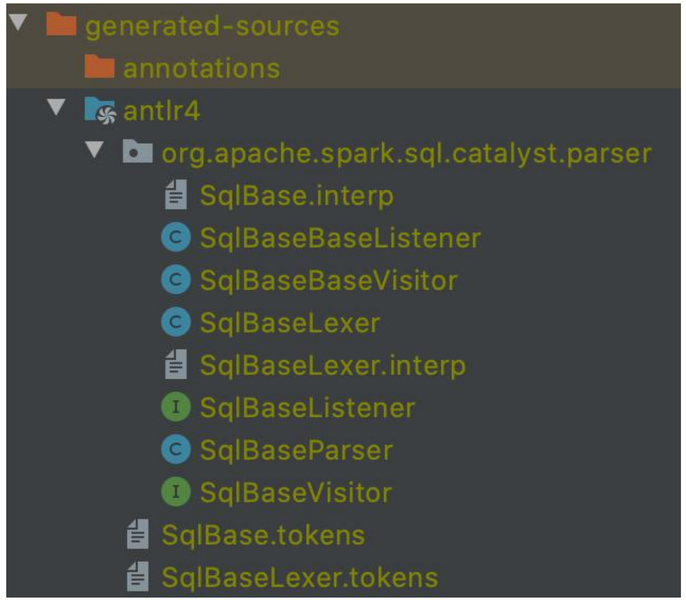
Unresolved Logical Plan

生成语法树之后，使用 AstBuilder 将语法树转换成 LogicalPlan，这个 LogicalPlan 也被称为 Unresolved LogicalPlan。解析后的逻辑计划如下：



SqlBase.g4

```
statement
    : query                                                          #statementDefault
    | ctes? dmlStatementNoWith                                       #dmlStatement
    | USE NAMESPACE? multipartIdentifier                             #use
    | CREATE namespace (IF NOT EXISTS)? multipartIdentifier
        (commentSpec |
         locationSpec |
         (WITH (DBPROPERTIES | PROPERTIES) tablePropertyList))*       #createNamespace
    | CREATE VIEW DATABASE (IF NOT EXISTS)? identifier ON identifierList
        (COMMENT comment=STRING)? locationSpec?
        (WITH DBPROPERTIES tablePropertyList)?                       #createViewDatabase
    | ALTER namespace multipartIdentifier
        SET (DBPROPERTIES | PROPERTIES) tablePropertyList            #setNamespaceProperties
    | ALTER namespace multipartIdentifier
        SET locationSpec                                             #setNamespaceLocation
    | DROP namespace (IF EXISTS)? multipartIdentifier
        (RESTRICT | CASCADE)?                                        #dropNamespace
    | SHOW (DATABASES | NAMESPACES) ((FROM | IN) multipartIdentifier)?
        (LIKE? pattern=STRING)?                                      #showNamespaces
    | CREATE (TEMPORARY | VOLATILE) TABLE (IF NOT EXISTS)? multipartIdentifier
        ('(' colTypeList ')')?
        createTableClauses
        (AS? query)?                                                 #createTable
    | createTableHeader ('(' colTypeList ')')? tableProvider
        createTableClauses
        (AS? query)?                                                 #createTable
    | createTableHeader ('(' columns=colTypeList ')')?
        (commentSpec |
        (PARTITIONED BY '(' partitionColumns=colTypeList ')' |
        PARTITIONED BY partitionColumnNames=identifierList) |
        bucketSpec |
        skewSpec |
        rowFormat |
        createFileFormat |
        locationSpec |
        (TBLPROPERTIES tableProps=tablePropertyList))*
        (AS? query)?                                                 #createHiveTable
```



SparkSqlParser

```scala
/**
 * Concrete parser for Spark SQL statements.
 */
class SparkSqlParser extends AbstractSqlParser {
  val astBuilder = new SparkSqlAstBuilder()

  private val substitutor = new VariableSubstitution()

  protected overri
    super.parse(su
  }
}

/**
 * Builder that co
 */
class SparkSqlAstBu
  import org.apache
```

```scala
override def visitCreateIndex(ctx: CreateIndexContext): LogicalPlan = withOrigin(ctx) {
  val deferredRebuild = ctx.DEFERRED != null
  val index = CatalogIndex(
    visitTableIdentifier(ctx.tableName),
    ctx.identifier().getText,
    string(ctx.indexType),
    visitIdentifierList(ctx.identifierList()),
    null,
    IndexReadOptions.VERSION,
    deferredRebuild, // deferredRebuild must be true
    Option(Option(ctx.comment).map(string).orNull)
  )
  CreateIndexCommand(index)
}
```

Catalog 和 HiveCatalog

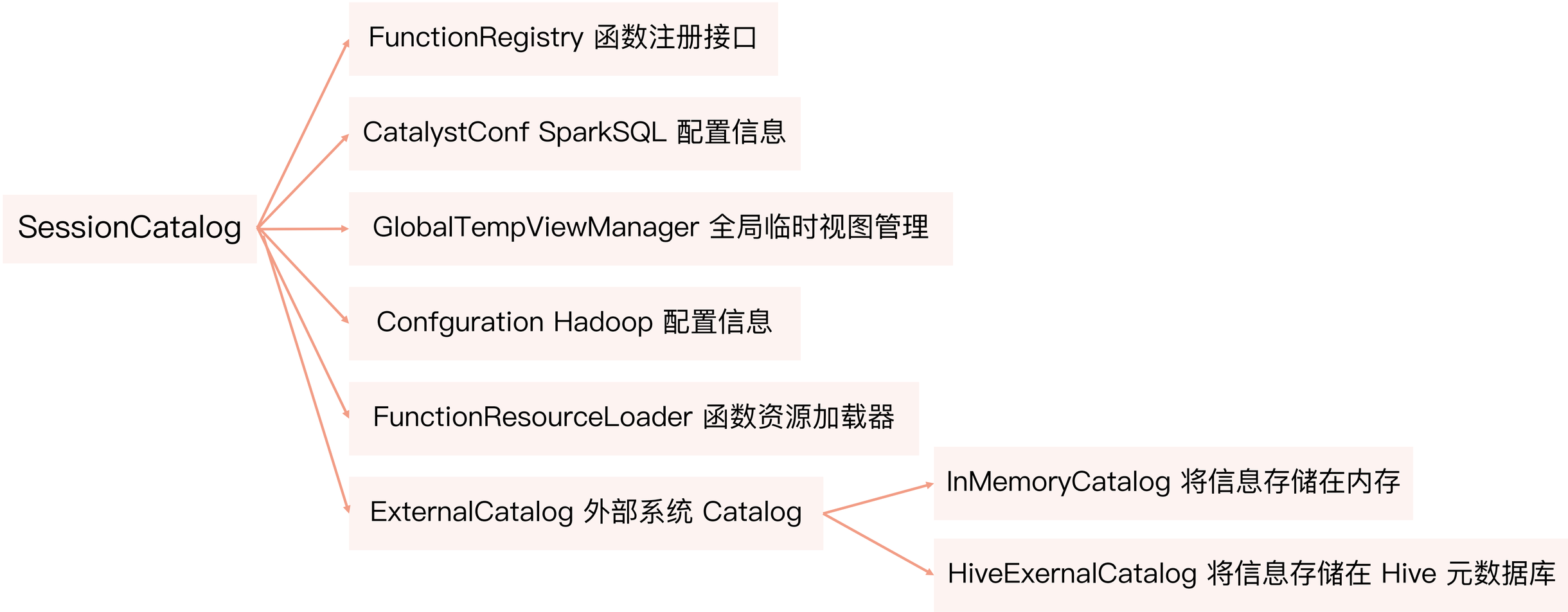
数据绑定

- Unresolved LogicalPlan 仅仅是一种数据结构，不包含任何数据信息，比如不知道数据源、数据类型、不同的列来自于哪张表等。接下来我们需要对其进行“数据绑定”。
- 数据绑定需要用到 Catalog。
- Catalog 是一种数据库用语。 在英文原意中为编目的意思，但是在数据库中完全可以不用翻译。

Catalog

- Catalog 主要用于各种函数资源信息和元数据信息（数据库、数据表、数据视图、数据分区与函数等）的统一管理。
- Spark 的 Catalog 管理类叫 SessionCatalog，此类管理着临时表、view、函数及外部依赖元数据（如 hive metastore），是 Analyzer 进行绑定的桥梁。

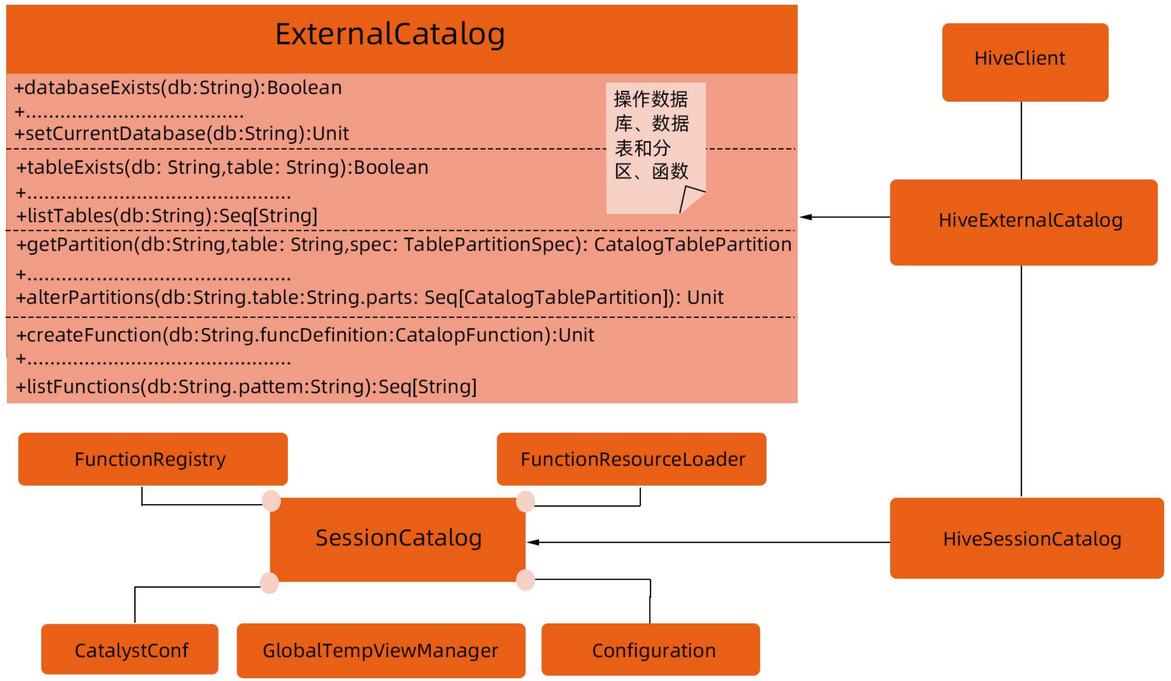
Catalog


SessionCatalog
FunctionRegistry 函数注册接口
CatalystConf SparkSQL 配置信息
GlobalTempViewManager 全局临时视图管理
Confguration Hadoop 配置信息
FunctionResourceLoader 函数资源加载器
ExternalCatalog 外部系统 Catalog
InMemoryCatalog 将信息存储在内存
HiveExernalCatalog 将信息存储在 Hive 元数据库

HiveExternalCatalog

SessionCatalog 实际上只是接口，具体的 Catalog 实现为 ExternalCatalog，目前有两种实现，一个是 InMemoryCatalog，一个是 HiveExternalCatalog。

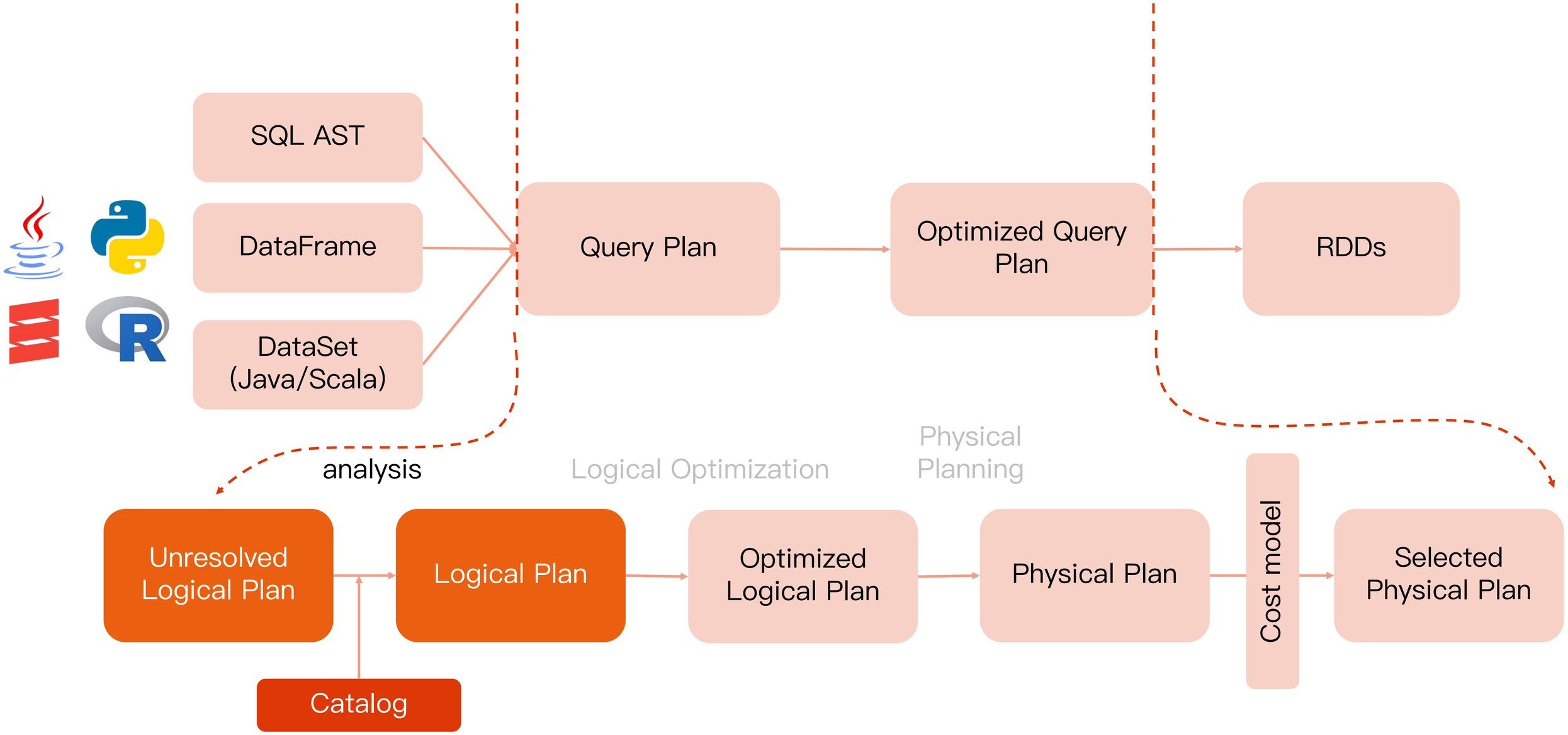
前者用于测试，而生产环境一般用后者： HiveExternalCatalog 是实际存储与操作数据的类。



Analyzer

1. sql text 经过 SqlParser 解析成 Unresolved LogicalPlan。
2. Analyzer 模块结合 Catalog 进行绑定，生成 Resolved LogicalPlan。

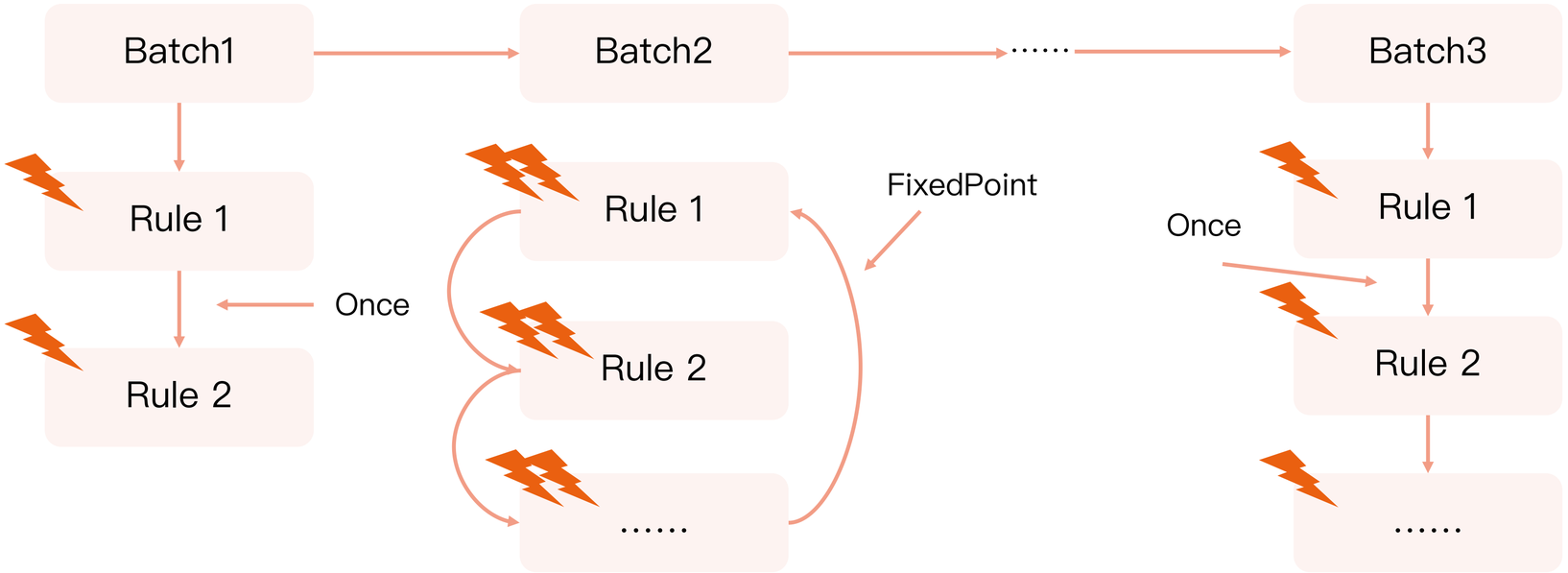
Analyzer


SQL AST
DataFrame
DataSet
(Java/Scala)
Query Plan
Optimized Query
Plan
RDDs
analysis
Logical Optimization
Physical
Planning
Unresolved
Logical Plan
Logical Plan
Optimized
Logical Plan
Physical Plan
Cost model
Selected
Physical Plan
Catalog

那么 Analyzer 是如何结合 Catalog 的呢?

那么 Analyzer 是如何结合 Catalog 的呢?

- Analyzer 会使用事先定义好的 Rule 以及 SessionCatalog 对 Unresolved LogicalPlan 进行 transform 操作。
- Rule 是定义在 Analyzer 里面的。
- 多个性质类似的 Rule 组成一个 Batch，多个 Batch 构成一个 Batches。
- Batches 由 RuleExecutor 执行，执行顺序如下图：



Rule 举例

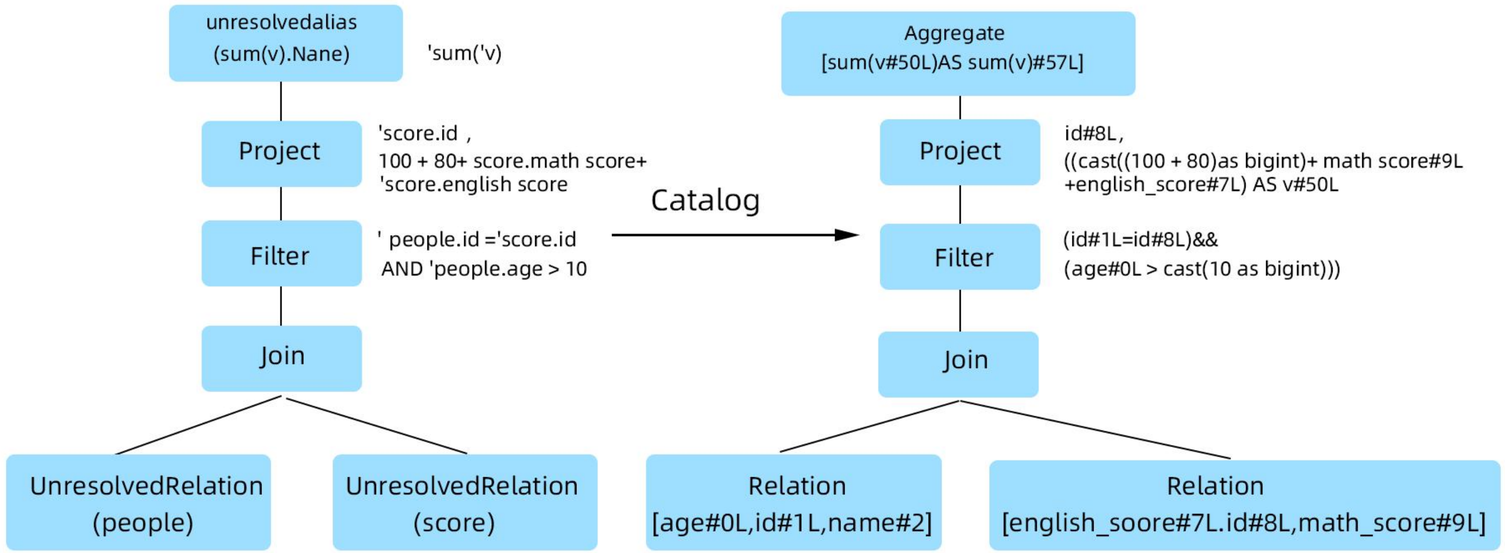
Rule 的种类有非常多，之前所说的和 SessionCatalog 结合一起对数据库、表、列的解析就是其中的几种 Rule（ResolveRelations，ResolveReference等）。

```
Batch ("Resolution", fixedPoint,
      ResolveTableValuedFunctions : :                  / / 解析表的函数
      ResolveRelations : :                               / / 解析表或视图
      ResolveReferences : :                             / / 解析列
      ResolveCreateNamedStruct : :
      ResolveDeserializer : :                          / / 解析反序列化操作
类
      ResolveNewInstance : :
      ResolveUpCast : :                                  / / 解析类型转换
      ResolveGroupingAnalytics : :
```

Rule 举例

解析规则	转换操作
ResolveTableValuedFunctions	解析可以作为数据表的函数（例如 range）
ResolveRelations	解析数据表
ResolveReferences	解析列
ResolveCreateNamedStruct	解析结构体创建
ResolveDeserializer	解析反序列化操作类
ResolveNewInstance	解析新的实例
ResolveUpCast	解析类型转换
ResolveGroupingAnalytics	解析多维分析
ResolvePivot	解析 Pivot
ResolveOrdinalInOrderByAndGroupBy	解析下标聚合
ResolveMissingReferences	解析新的列
ExtractGenerator	解析生成器
ResolveGenerate	解析生成过程
ResolveFunctions	解析函数
ResolveAliases	解析别名
ResolveSubquery	解析子查询
ResolveWindowOrder	解析窗口函数排序
ResolveWindowFrame	解析窗口函数
ResolveNaturalAndUsingJoin	解析自然J oin
ExtractWindowExpressions	提取窗口函数表达式
GlobalAggregates	解析全局聚合
ResolveAggregateFunctions	解析聚合函数
TimeWindowing	解析时间窗口
ResolveInlineTables	解析内联表
TypeCoercion.typeCoercionRules	解析强制类型转换
extendedResolutionRules	扩展规则

Resolved Logical Plan



16/12/2613:18:48DEBUG AnalyzerSRcsolveReferences: Resolvingpcoplc.id to id# 1 L
16/12/2613:18:48 DEBUG AnalyzerSResolveReferences:Resolving'score.id to id#8L
16/12/2613:18:48 DEBUG AnalyzerSResolveReferences: Resolving people.age to agen# 0 L
16/12/26 13:18:48 DEBUG AnalyzerSRcsolveReferences:Resolving'score.id to id#8l
16/12/2613:18:48DEBUG AnalyzcrSResolveRefcrences: Resolving 'score.math score to math scorc#9L
16/12/2613:18:48DEBUG AnalyzerSResolveReferences: Resolving 'scoreenglish score to english scoren # 7 L
16/12/2613:18:48 DEBUG AnalyzerSResolveReferences:Resolving'v to v # 50L

THANKS