Apache Spark 2.0 Catalyst Engine

Preliminary Notes

By Oliver Tupran @ Lunatech 2016

Intro

- Catalyst is a query optimisation engine
- The Catalyst engine was introduced in 2014 in Spark 1.0.0 (see SPARK-1251)

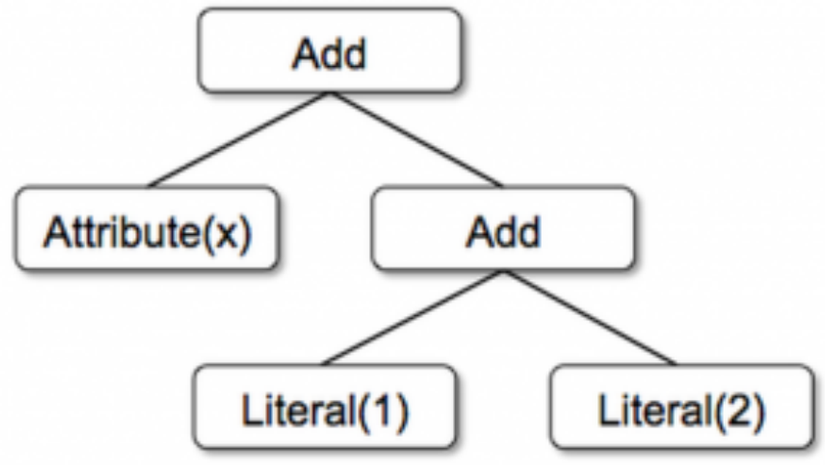
Personal code experiments on GitHub

Trees

- `TreeNode` is the main data type in Catalyst
- `TreeNodes` are immutable
- `TreeNodes` can be manipulated through recursive functional transformations (e.g. map, flatMap, foreach, transform…)
- `TreeNode` has two main implementations: `Expression` and `QueryPlan`

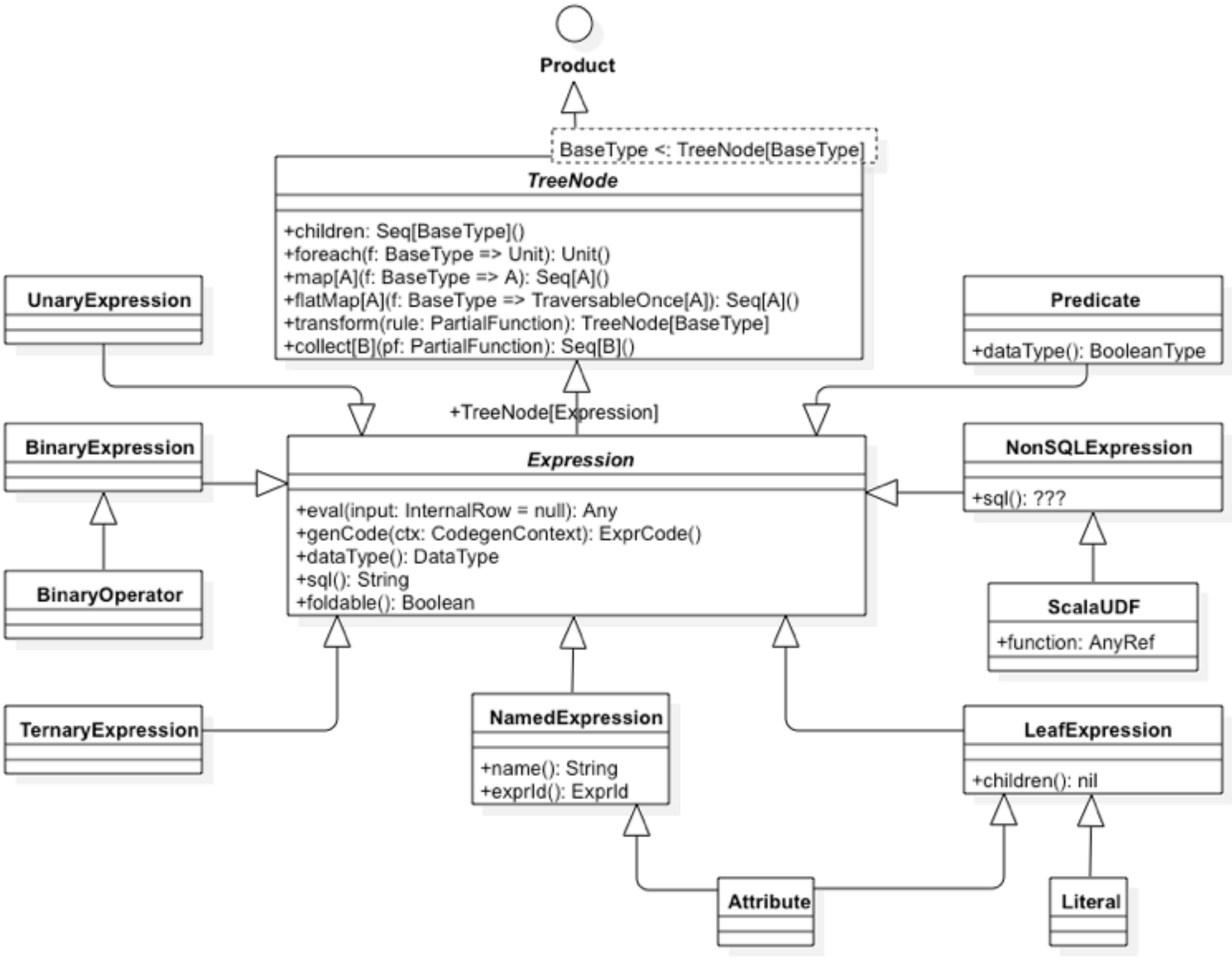
Expression Example

```
x + (1 + 2)
```




```
Add(Attribute(“x”), Add(Literal(1), Literal(2)))
```

Expressions



Rules

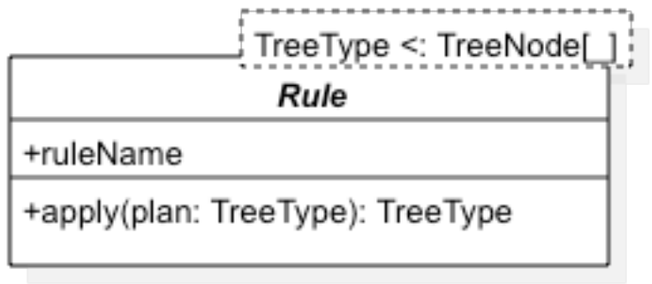
- `Rules` are functions, or partial functions, that transform a `TreeNode` into a `TreeNode` (though `Rule` does not extend `Function1`)

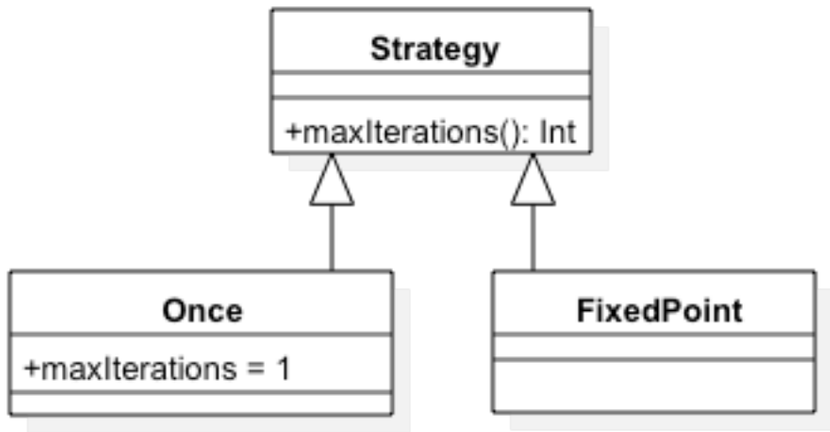
- `Rules` are applied using `RuleExecutors`

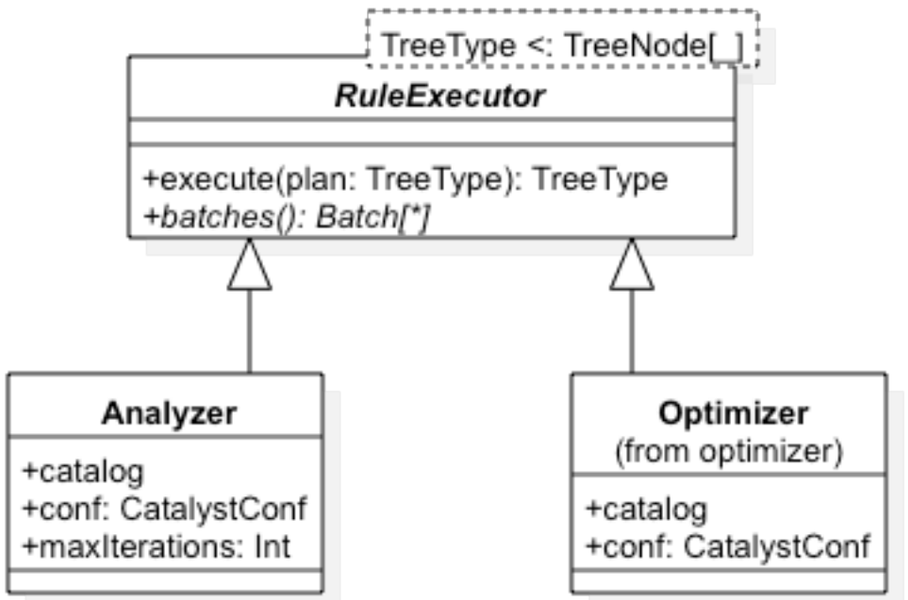
- `RuleExecutors` apply batches of `Rules` to the `TreeNode` using various strategies (most common are `Once` or `FixedPoint`)

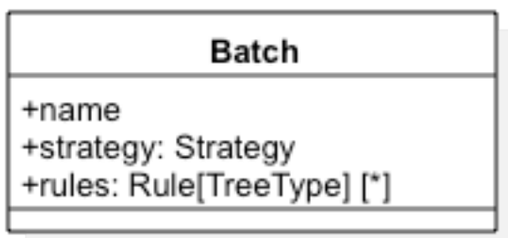
- Two main implementations of `RuleExecutor` are `Analyser` and `Optimizer`

Rules

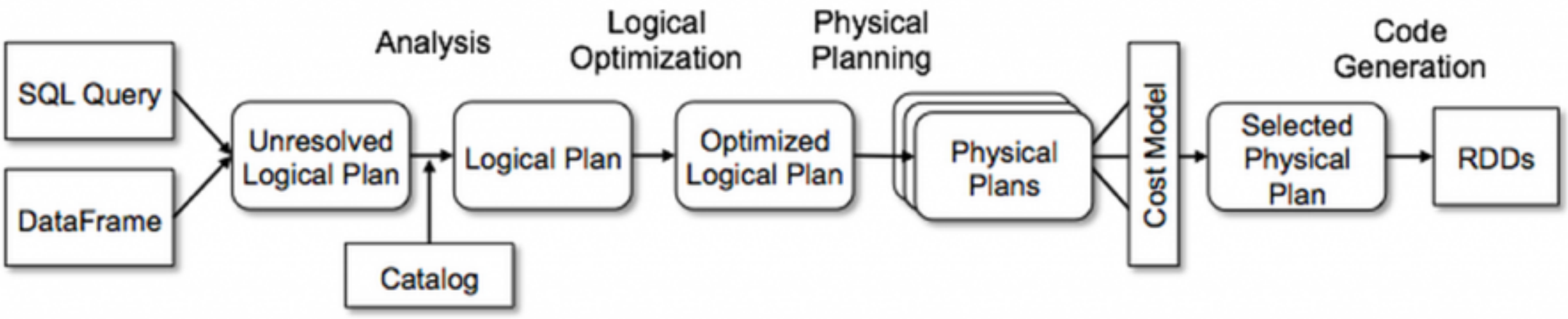

TreeType <: TreeNode[_]
Rule
+ruleName
+apply(plan: TreeType): TreeType


Strategy
+maxIterations(): Int
Once
+maxIterations = 1
FixedPoint


TreeType <: TreeNode[_]
RuleExecutor
+execute(plan: TreeType): TreeType
+batches(): Batch[*]
Analyzer
+catalog
+conf: CatalystConf
+maxIterations: Int
Optimizer
(from optimizer)
+catalog
+conf: CatalystConf


Batch
+name
+strategy: Strategy
+rules: Rule[TreeType] [*]

Catalyst Work Flow

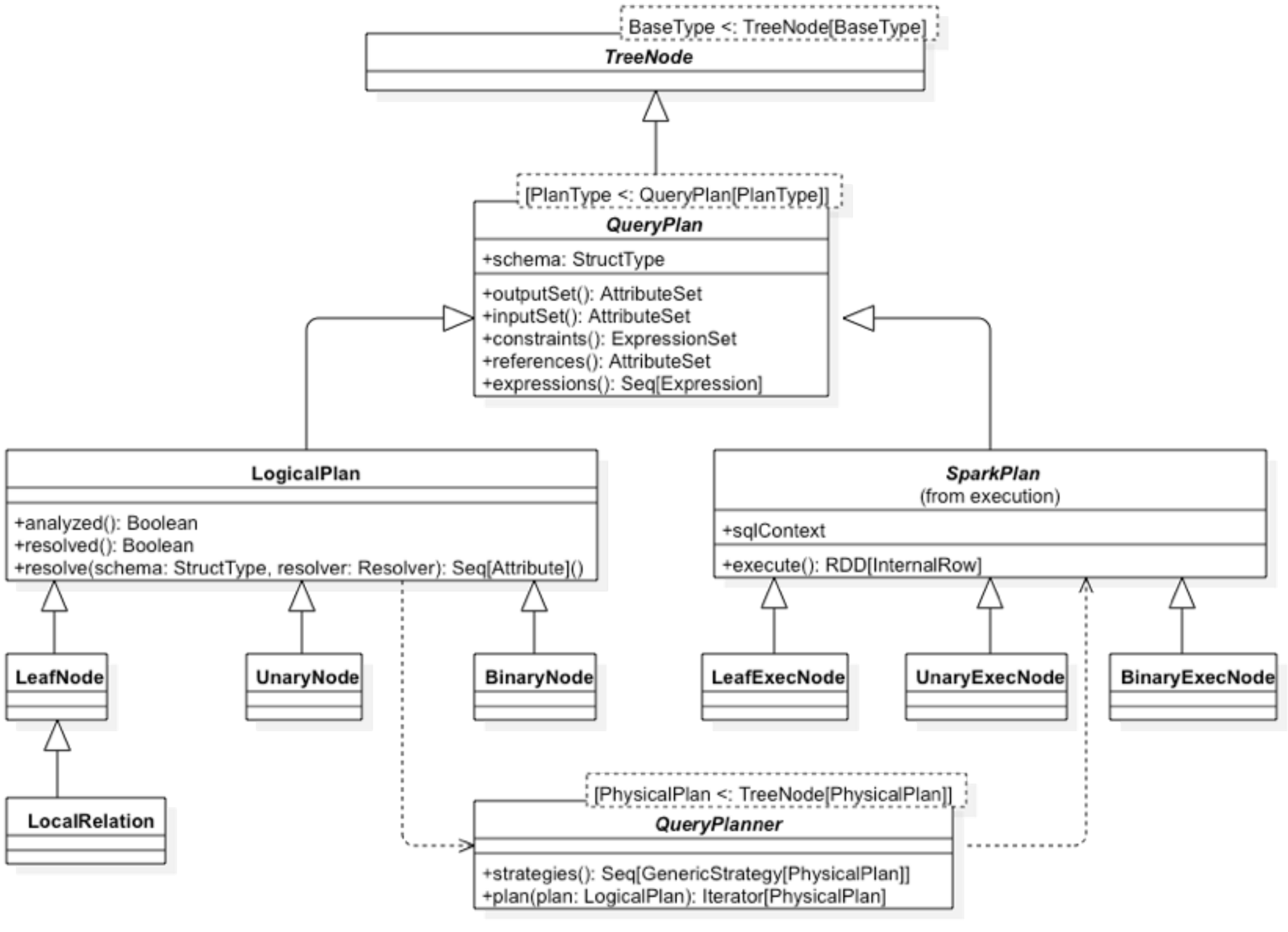


Analysis

- Transform raw, unresolved `LogicalPlan` into a resolved and analysed `LogicalPlan`
- Steps
  - resolve relations (tables)
  - resolve attributes
  - resolving attributes referring to the same value
  - resolve attribute types and casts

See Analyzer

Query Plans


BaseType <: TreeNode[BaseType]
TreeNode
[PlanType <: QueryPlan[PlanType]]
QueryPlan
+schema: StructType
+outputSet(): AttributeSet
+inputSet(): AttributeSet
+constraints(): ExpressionSet
+references(): AttributeSet
+expressions(): Seq[Expression]
LogicalPlan
+analyzed(): Boolean
+resolved(): Boolean
+resolve(schema: StructType, resolver: Resolver): Seq[Attribute]()
SparkPlan
(from execution)
+sqlContext
+execute(): RDD[InternalRow]
LeafNode
UnaryNode
BinaryNode
LeafExecNode
UnaryExecNode
BinaryExecNode
LocalRelation
[PhysicalPlan <: TreeNode[PhysicalPlan]]
QueryPlanner
+strategies(): Seq[GenericStrategy[PhysicalPlan]]
+plan(plan: LogicalPlan): Iterator[PhysicalPlan]

Logical Optimisation

- The logical optimiser is producing a new logical plan
- Examples:
  - constant folding
  - predicate pushdown
  - projection pruning
  - null propagation
  - boolean expression simplification

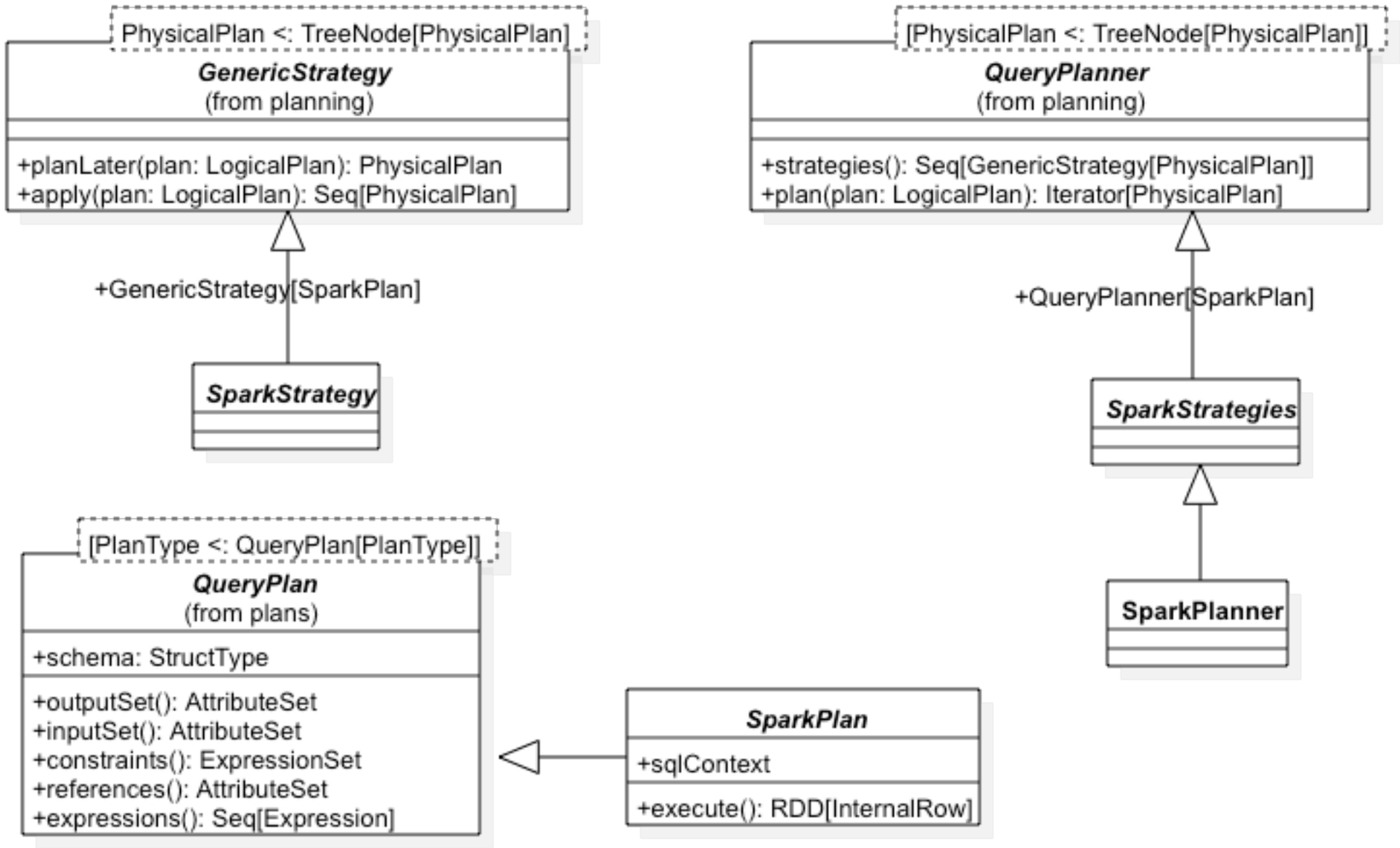
See Optimizer

Physical Planning

- From a `LogicalPlan` produce one or more `PhysicalPlans`
- On the resulted `PhysicalPlans` both cost-based and rules-based optimisations are applied
- If possible it will do a Projection Push-Down, to the external system (e.g. HDFS, Cassandra, RDMS…)
- The resulted `PhysicalPlan` is optimised for Spark

See SparkStrategies

Execution


PhysicalPlan <: TreeNode[PhysicalPlan]
GenericStrategy
(from planning)
+planLater(plan: LogicalPlan): PhysicalPlan
+apply(plan: LogicalPlan): Seq[PhysicalPlan]
+GenericStrategy[SparkPlan]
SparkStrategy
[PhysicalPlan <: TreeNode[PhysicalPlan]]
QueryPlanner
(from planning)
+strategies(): Seq[GenericStrategy[PhysicalPlan]]
+plan(plan: LogicalPlan): Iterator[PhysicalPlan]
+QueryPlanner[SparkPlan]
SparkStrategies
SparkPlanner
[PlanType <: QueryPlan[PlanType]]
QueryPlan
(from plans)
+schema: StructType
+outputSet(): AttributeSet
+inputSet(): AttributeSet
+constraints(): ExpressionSet
+references(): AttributeSet
+expressions(): Seq[Expression]
SparkPlan
+sqlContext
+execute(): RDD[InternalRow]

Code Generation

- Mainly based on Scala Quasiquotes, but there is some string based generation as well
- No virtual function calls, linear code
- Entire code fits in one function => data passes through CPU registers vs memory

See CodeGenerator

TODOs

- Find if Catalyst truly needs a dependency to spark-core
- Find the best way to deliver the code:
  - source or
  - byte code or
  - logical plan
- Test more complex plans

References

- Spark SQL: Relational Data Processing in Spark
- SPARK-1251 Catalyst
- SPARK-12795 Whole stage codegen
- Deep Dive into Spark SQL's Catalyst Optimizer
- Apache Spark as a Compiler
- anatomy-of-spark-catalyst