openLooKeng新下推框架介绍和实践

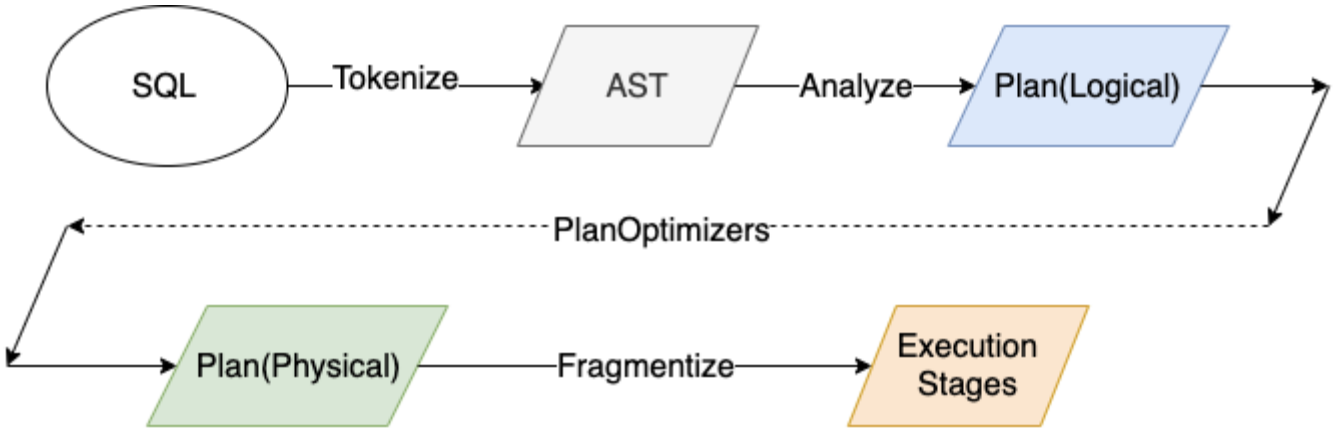
罗旦

openLooKeng

https://openlookeng.io

目录

Planner/Optimizer



- Presto executes a SQL query by first parsing it to an Abstract Syntax Tree (AST). The AST is then converted to logical plan tree, which represents the relational algebra contained in the query. The relational algebra representation is not optimized and lacks sufficient physical layout information for query execution.

- Presto uses a list of optimizers to transform the logical plan to an optimized physical plan. Each plan optimizer can operate on sub-trees of the whole plan tree and replace them with more optimized sub-trees based on heuristic or statistics. Optimizers can save the physical information of execution on a set of connector provided handles (for example, ConnectorTableHandle, ConnectorTableLayoutHandle, ConnectorPartitionHandle, …).

openLooKeng

PrestoSql Pushdown Solution

Goals and non-goals:

- In the long term, we want connectors to be able to provide transformation rules that get evaluated during the optimization loop.
- It should be implemented using the existing Rule/Iterative optimizer framework instead of visitor-based PlanOptimizers.
- It's not a goal to build a generic mechanism to support every operation.

Pushdown Solution:

- The general idea is to introduce a new set of transformation rules, each of which fires on patterns such as filter(tableScan(...)), project(tableScan(...)), etc. Each rule would be responsible for pushing down the corresponding type of operation into the table scan. Examples would be: PushFilterIntoConnector, PushProjectionIntoConnector, PushAggregationIntoConnector, PushJoinIntoConnector.

- These rules interact with connectors via a set of specialized metadata calls. If a connector doesn't support or understand the action, result of the call would indicate so. The language to express filters, projections, aggregates, join criteria, etc. is also TBD.

PrestoSql Solution: Filter pushdown

Example 1

Let's say a connector knows how to handle LIKE expressions. Given the following plan fragment,

```
- Filter (a like 'xy%z%' AND f(b)) :: (a varchar, b bigint)
  - TableScan (TH(0)) :: (a varchar, b bigint)
    a = CH(0)
    b = CH(1)
```

`PushFilterIntoTableScan` would call:

```
Metadata.pushFilter(
  TH(0),
  call("and",
      call("like", CH(0), 'xy%z%'),
      call("f", CH(1))))
```

HiveConnector.pushFilter

which would return

```
new table handle: TH(0')
remaining filter: call("f", CH(1))
```

The rule would then replace the original fragment with:

```
- Filter (f(b)) :: (a varchar, b bigint)
  - TableScan (TH(0')) :: (a varchar, b bigint)
    a = CH(0)
    b = CH(1)
```

Metadata.pushFilter(TableHandle, Filter) => TableHandle + remaining filter + new projections.

Returns a new table handle and any part of the filter that the connector doesn't understand or support. The result TableHandle represents a table with the same schema as the input TableHandle.

The optimization rule uses this method to transform filter(f1, tablescan(th1)) into filter(f2, tablescan(th2)).

openLooKeng

PrestoSql Solution: Aggregation pushdown

Example

Given this initial plan fragment:

```
- Aggregation[SingleStep] :: (k varchar, x bigint, y double)
    group by: [k]
    aggregates: [x = sum(a), y = avg(b)]
  - TableScan(TH(0)) :: (k varchar, a bigint, b bigint)
     k = CH(0)
     a = CH(1)
     b = CH(2)
```

The rule calls:

```
Metadata.pushAggregation(
  TH(0),
  false
  [CH(0)],
  [
    aggregate("sum", CH(1)),
    aggregate("avg", CH(2))
  ])
```

which returns:

```
new table handle: TH(0')
new column handles for aggregates: [CH(3), CH(4)]
```

The fragment is rewritten to:

```
- TableScan(TH(0')) :: (k varchar, x bigint, y double)
   k = CH(0)
   x = CH(3)
   z = CH(4)
```

Metadata.pushAggregation (TableHandle, partial?, <group-by columns>, <aggregates>) => TableHandle + new column handles for aggregates

PrestoSql当前方案的问题：

- 很难处理复杂sql的下推，比如Join Node，Window Node，特别是多表join
- 下推的上下文信息处理复杂
- 处理下推的逻辑复杂

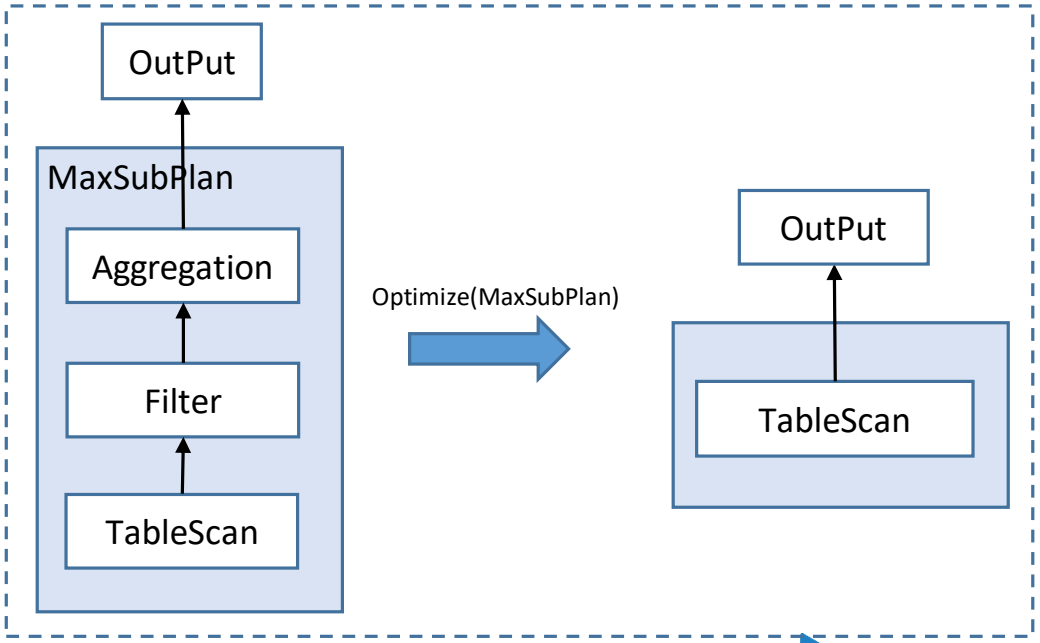
openLooKeng

openLooKeng solution: Exposing plan sub-trees to connector

- openLooKeng now allows connectors to provide optimization rules to the Presto engine, which allows connectors to introduce arbitrary optimizations. There are restrictions to prevent connector provided optimizers from accidentally changing another connector's sub plan:
  - PlanNodes that are exposed to presto-spi module.
  - PlanNodes that belong to the connector.

A sub max tree that satisfies the above rules will be transformed to more optimized form picked by a connector provided optimization rule.

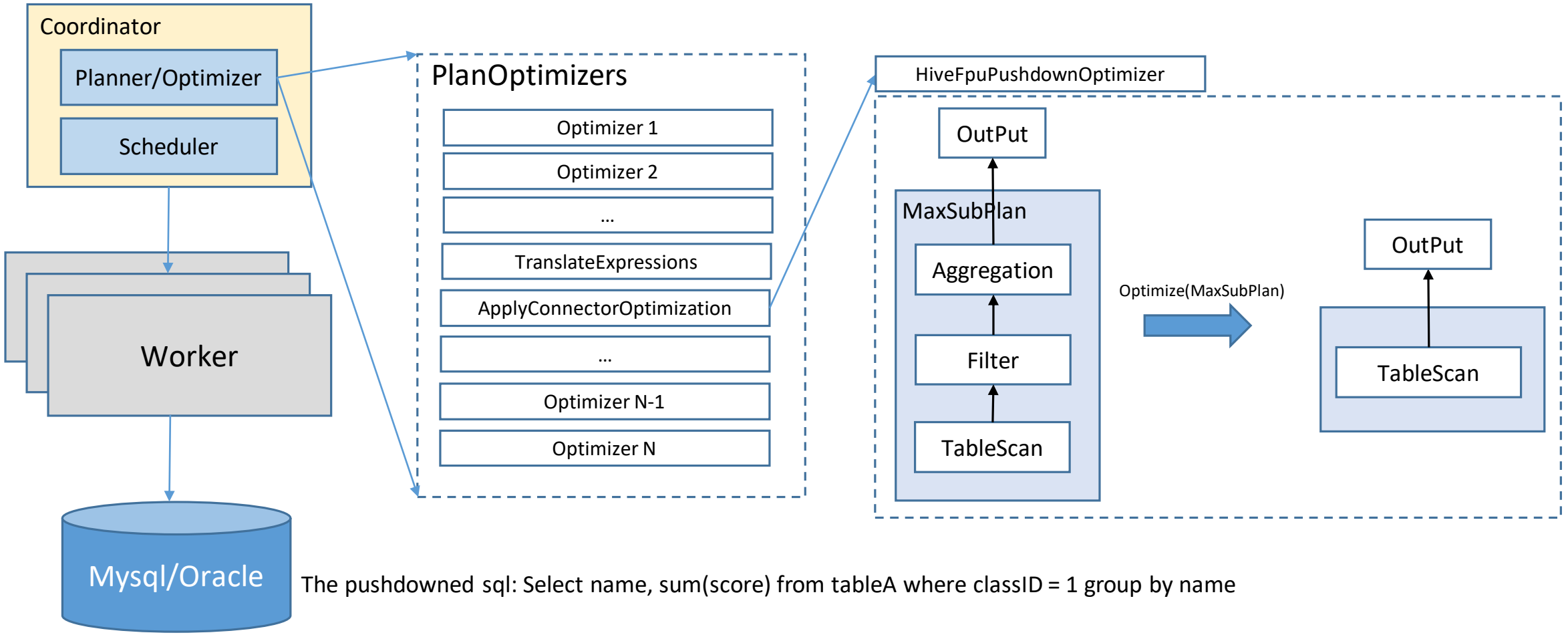
```
public  interface  ConnectorPlanOptimizer
{
    PlanNode optimize(
        PlanNode maxSubplan,
        ConnectorSession session,
        VariableAllocator variableAllocator,
        PlanNodeIdAllocator idAllocator);
}
```



openLooKeng

openLooKeng solution: Pushdown Architecture

Select name, sum(score) from tableA where classID = 1 group by name



The pushdowned sql: Select name, sum(score) from tableA where classID = 1 group by name

- **TranslateExpressions optimizer is used to transform the Expression to RowExpression in PlanNode.**
- **ApplyConnectorOptimization optimizer applies ConnectorPlanOptimizers to pushdown the operation into data source.**

openLooKeng

Modification

PR—https://gitee.com/openlookeng/hetu-core/pulls/633

- Move some PlanNodes to presto-spi module

- Change Expression to RowExpression in PlanNode
- Change Expression to RowExpression in Assignments

- Modify rules and Optimizers

- Add TranslateExpressions and ApplyConnectorOptimization Optimizers

- Add ConnectorPlanOptimizers for connector

Expression-to-RowExpression

Detaching AST (Node) from IR (PlanNode) has been discussed for years in the community. Especially, we will replace Expression references in PlanNode with RowExperssion.

The current lifecycle a plan (before being compiled as operators) is:

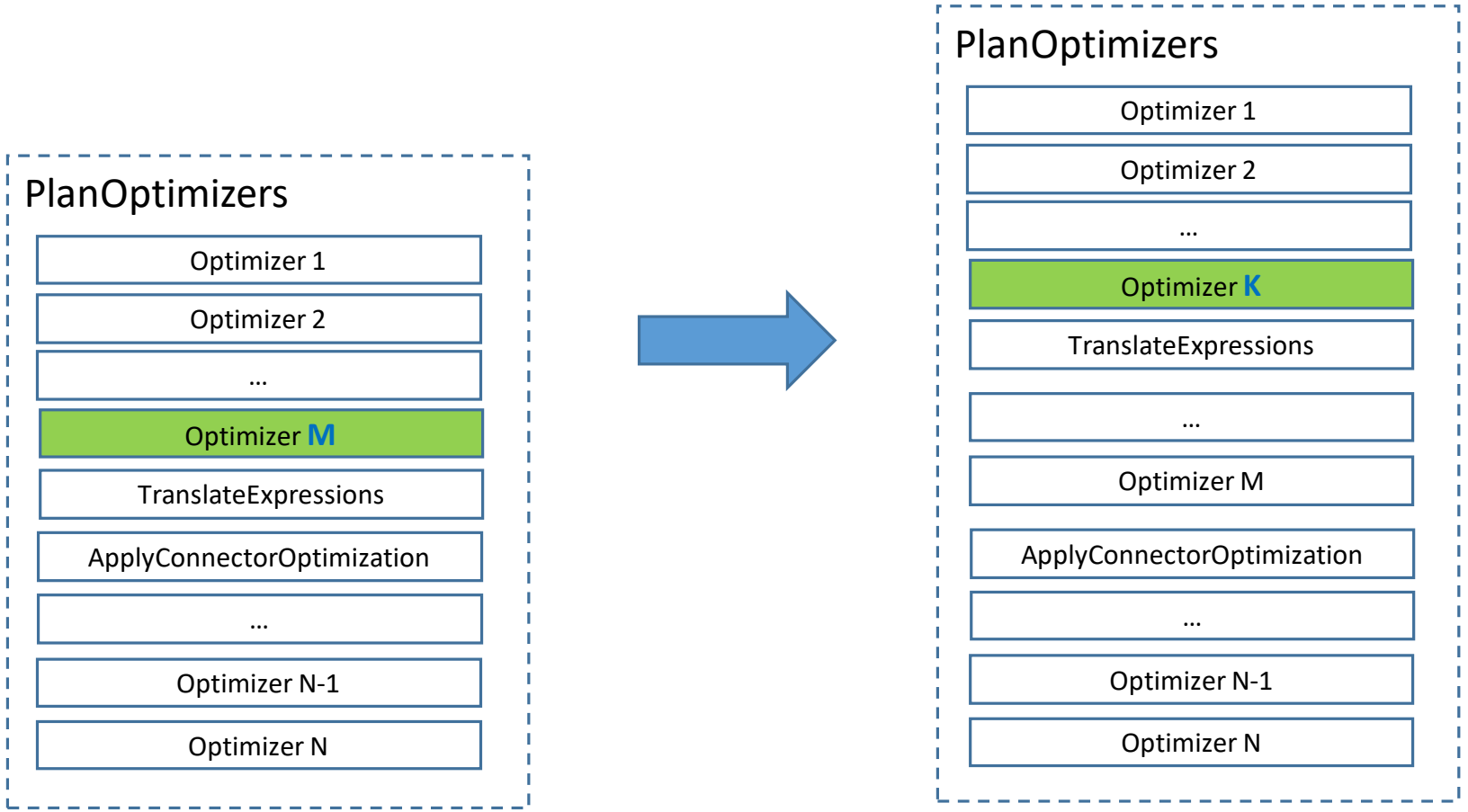
1. building AST
2. building raw plan
3. plan optimization
4. plan sanity check
5. plan cost computation
6. building subplans
7. distributing subplan (over the wire)
8. compiling subplan locally

Expression-to-RowExpression translation happens at step 8 as of today. We are moving it to step 3 (and future to step 2). The reason to put the translation at step 3 instead of step 2 is due to heavy references of Expression in optimizers.

openLooKeng

The future work

- Move up TranslateExpressions Optimizer (K < M) (PR 840)



openLooKeng

Example 演示

```
lk:tpcds> explain select c_first_name,c_last_name,ca_city,ca_county from customer join customer_address on c_current_addr_sk=ca_address_sk where c_customer_sk=1;

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Output[c_first_name, c_last_name, ca_city, ca_county]
 │   Layout: [c_first_name:char(20), c_last_name:char(30), ca_city:varchar(255), ca_county:varchar(255)]
 │   Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: ?}
 └─ RemoteExchange[GATHER]
    │   Layout: [c_first_name:char(20), c_last_name:char(30), ca_city:varchar(255), ca_county:varchar(255)]
    │   Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: ?}
    └─ TableScan[table = mysql:mysql.GeneratedSql{sql=SELECT c_first_name, c_last_name, ca_city, ca_county FROM ((SELECT c_current_addr_sk, c_first_name, c_last_na
       Layout: [c_first_name:char(20), c_last_name:char(30), ca_city:varchar(255), ca_county:varchar(255)]
       Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: 0B}
       ca_county := ca_county:varchar(255):Optional[VARCHAR]
       ca_city := ca_city:varchar(255):Optional[VARCHAR]
       c_last_name := c_last_name:char(30):Optional[CHAR]
       c_first_name := c_first_name:char(20):Optional[CHAR]

(1 row)
```

openLooKeng

How to contribute（1）

第一：在XXXConnector中复写下面的函数

```
@Override
public ConnectorPlanOptimizerProvider getConnectorPlanOptimizerProvider()
{
    return new DruidPlanOptimizerProvider(planOptimizer);
}
```

第二：实现XXXPlanOptimizerProvider

```
public class DruidPlanOptimizerProvider
        implements ConnectorPlanOptimizerProvider
{
    private final ConnectorPlanOptimizer planOptimizer;

    @Inject
    public DruidPlanOptimizerProvider(
            ConnectorPlanOptimizer planOptimizer)
    {
        this.planOptimizer = requireNonNull(planOptimizer, message: "planOptimizer is null");
    }

    @Override
    public Set<ConnectorPlanOptimizer> getLogicalPlanOptimizers() { return ImmutableSet.of(planOptimizer)

    @Override
    public Set<ConnectorPlanOptimizer> getPhysicalPlanOptimizers() { return ImmutableSet.of(); }
}
```

第三：在XXXConnector中实现PlanOptimizer

```
public class DruidPlanOptimizer
        implements ConnectorPlanOptimizer
{
    private final DruidQueryGenerator druidQueryGenerator;
    private final TypeManager typeManager;
    private final FunctionMetadataManager functionMetadataManager;
    private final LogicalRowExpressions logicalRowExpressions;
    private final StandardFunctionResolution standardFunctionResolution;

    @Inject
    public DruidPlanOptimizer(
            DruidQueryGenerator druidQueryGenerator,
            TypeManager typeManager,
            DeterminismEvaluator determinismEvaluator,
            FunctionMetadataManager functionMetadataManager,
            StandardFunctionResolution standardFunctionResolution)
    {
        this.druidQueryGenerator = requireNonNull(druidQueryGenerator, message: "pi
        this.typeManager = requireNonNull(typeManager, message: "type manager is nu
        this.functionMetadataManager = requireNonNull(functionMetadataManager, mes
        this.standardFunctionResolution = requireNonNull(standardFunctionResolutior
        this.logicalRowExpressions = new LogicalRowExpressions(
                determinismEvaluator,
                standardFunctionResolution,
                functionMetadataManager);
    }
```

openLooKeng

How to contribute（2）

第四：实现PlanOptimizer里面的optimize函数，主要是实现一个visitor，去visit执行计划树

```
@Override
public PlanNode optimize(PlanNode maxSubplan,
        ConnectorSession session,
        VariableAllocator variableAllocator,
        PlanNodeIdAllocator idAllocator)
{
    Map<PlanNodeId, TableScanNode> scanNodes = maxSubplan.accept(new TableFindingVisitor(), context: null);
    return maxSubplan.accept(new Visitor(scanNodes, session, idAllocator), context: null);
}
```

第五：实现Visitor，用来生成下推的语句，同时修改执行计划树

```
private class Visitor
        extends PlanVisitor<PlanNode, Void>
{
    private final PlanNodeIdAllocator idAllocator;
    private final ConnectorSession session;
    private final Map<PlanNodeId, TableScanNode> tableScanNodes;
    private final IdentityHashMap<FilterNode, Void> filtersSplitUp = new IdentityHashMap<>();

    public Visitor(Map<PlanNodeId, TableScanNode> tableScanNodes, ConnectorSession session, PlanNodeIdAllocator idAll
    {
        this.session = session;
        this.idAllocator = idAllocator;
        this.tableScanNodes = tableScanNodes;
        // Just making sure that the table exists
        tableScanNodes.forEach((key, value) -> getDruidTableHandle(value).get().getTableName());
    }

    private Optional<PlanNode> tryCreatingNewScanNode(PlanNode plan)
    {
        Optional<DruidQueryGenerator.DruidQueryGeneratorResult> dql = druidQueryGenerator.generate(plan, session);
        if (!dql.isPresent()) {
            return Optional.empty();
        }
        DruidQueryGeneratorContext context = dql.get().getContext();
        final PlanNodeId tableScanNodeId = context.getTableScanNodeId().orElseThrow(() -> new PrestoException(DRUID_
        if (!tableScanNodes.containsKey(tableScanNodeId)) {
            throw new PrestoException(DRUID_QUERY_GENERATOR_FAILURE, "Expected to find a druid table scan node");
        }
```

第六：实现XXXQueryGenerator，在XXXQueryGenerator中实现一个visitor用来把下推的信息记录到XXXQueryGeneratorContext，如果存在节点可以下推，则生成对应的sql

```
public class DruidQueryGenerator
{
    private static final Logger log = Logger.get(DruidQueryGenerator.class);
    private static final Map<String, String> UNARY_AGGREGATION_MAP = ImmutableMap.of(
            k1: "min", v1: "min",
            k2: "max", v2: "max",
            k3: "avg", v3: "avg",
            k4: "sum", v4: "sum",
            k5: "distinctcount", v5: "DISTINCTCOUNT");

    private final TypeManager typeManager;
    private final FunctionMetadataManager functionMetadataManager;
    private final StandardFunctionResolution standardFunctionResolution;
    private final DruidProjectExpressionConverter druidProjectExpressionConverter;

    @Inject
    public DruidQueryGenerator(
            TypeManager typeManager,
            FunctionMetadataManager functionMetadataManager,
            StandardFunctionResolution standardFunctionResolution)
    {
        this.typeManager = requireNonNull(typeManager, message: "type manager is null");
        this.functionMetadataManager = requireNonNull(functionMetadataManager, message: "functi
        this.standardFunctionResolution = requireNonNull(standardFunctionResolution, message: "
        this.druidProjectExpressionConverter = new DruidProjectExpressionConverter(typeManager,
    }
```

openLooKeng

开源软件供应链点亮计划-暑期2021

https://openlookeng.io/zh-cn/events/summer-2021.html

开源软件供应链点亮计划-暑期2021（openLooKeng）

openLooKeng | 2021.02-2021.11

暑期2021活动简介

开源软件供应链点亮计划 - 暑期 2021（以下简称 暑期 2021）是由中国科学院软件研究所与openEuler社区共同举办的一项面向高校学生的暑期活动，旨在鼓励在校学生积极参与开源软件的开发维护，促进优秀开源软件社区的蓬勃发展。我们将联合各大开源社区，针对重要开源软件的开发与维护提供项目，并向全球高校学生开放报名。

学生自由选择项目，与社区导师沟通实现方案并撰写项目计划书。被选中的学生将在社区导师指导下，按计划完成开发工作，并将成果贡献给社区。根据项目的难易程度和完成情况，参与者将获取由主办方发放的项目奖金。

openLooKeng

openLooKeng社区欢迎您

官方网站： https://openlookeng.io/

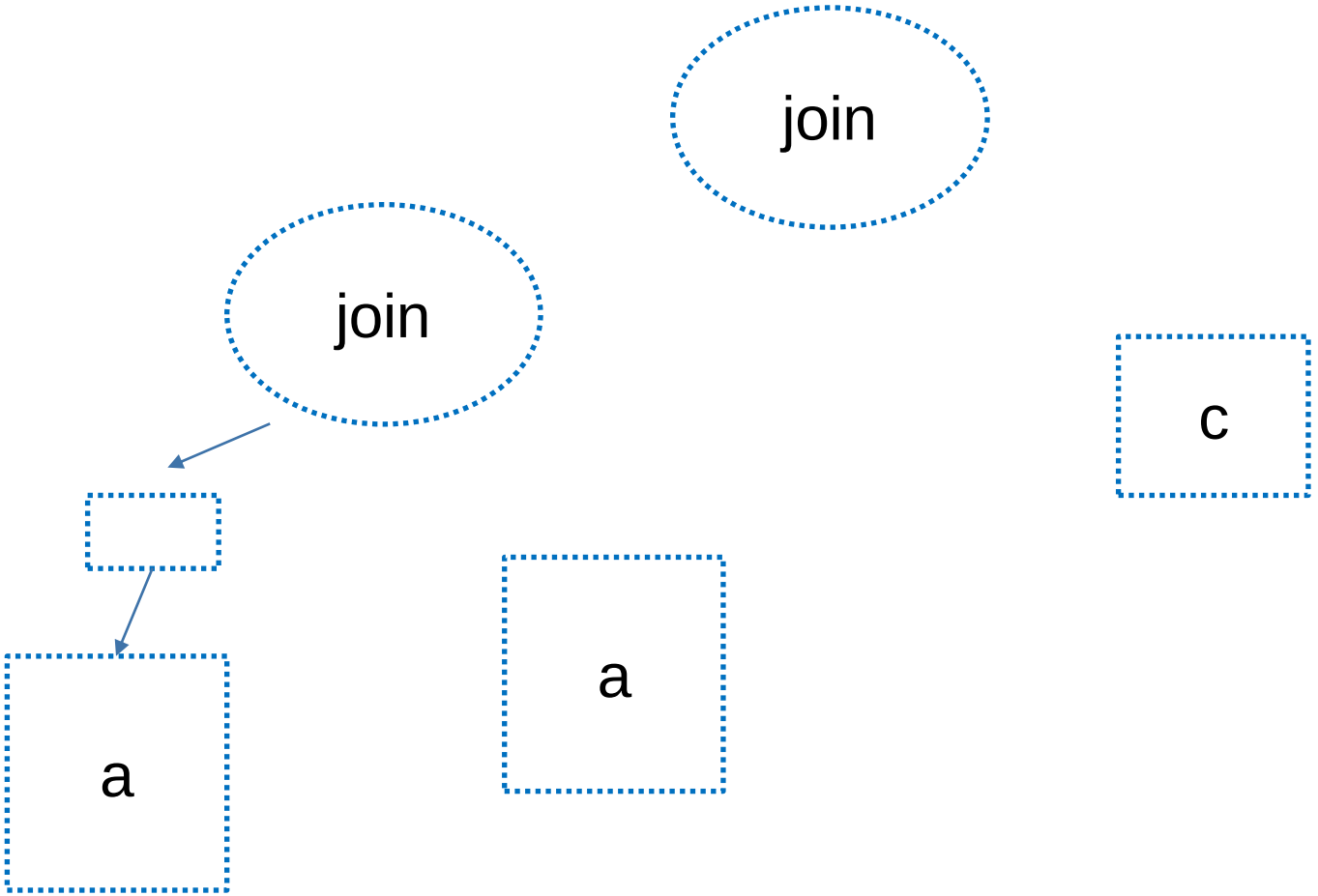
线上体验环境： https://tryme.openlookeng.io/

源代码开放： https://gitee.com/openlookeng/

openLooKeng微信公众号

openLooKeng微信小助手

openLooKeng



openLooKeng

Thank You.

openLooKeng

https://openlookeng.io