Apache Calcite

Julian Hyde | **AICamp** | 2018/10/16

@julianhyde

SQL
Query planning
Query federation
BI & OLAP
Streaming
Hadoop

ASF member
Original author of Apache Calcite
PMC Apache Arrow, Calcite, Drill, Eagle, Kylin
Architect at Looker

Apache Calcite

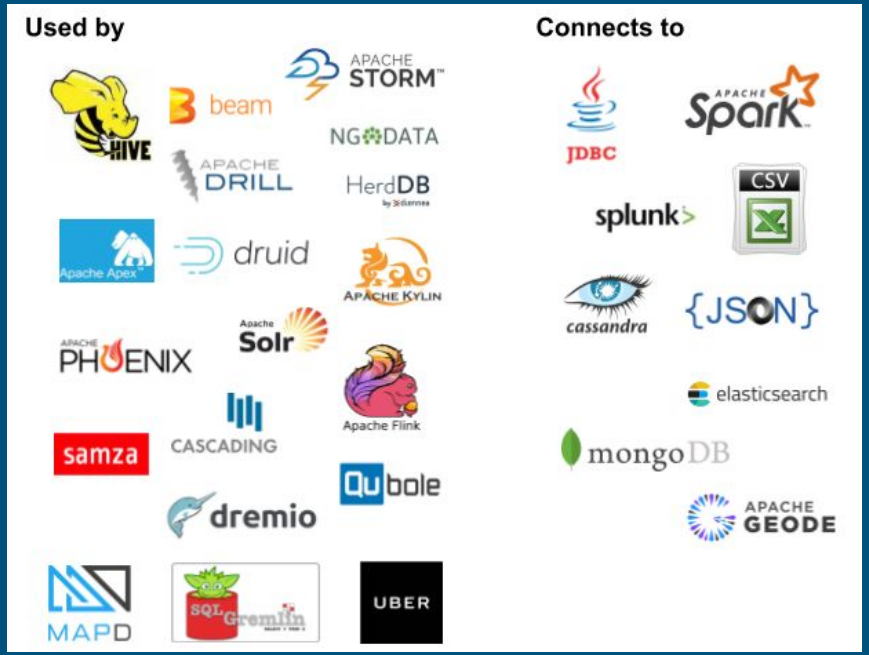
Apache top-level project

Query planning framework used in many projects and products

Also works standalone: embedded federated query engine with SQL / JDBC front end

Apache community development model

https://calcite.apache.org
https://github.com/apache/calcite



Project goals

Make it easier to write a simple DBMS

Advance the state of the art for complex DBMS by pooling resources

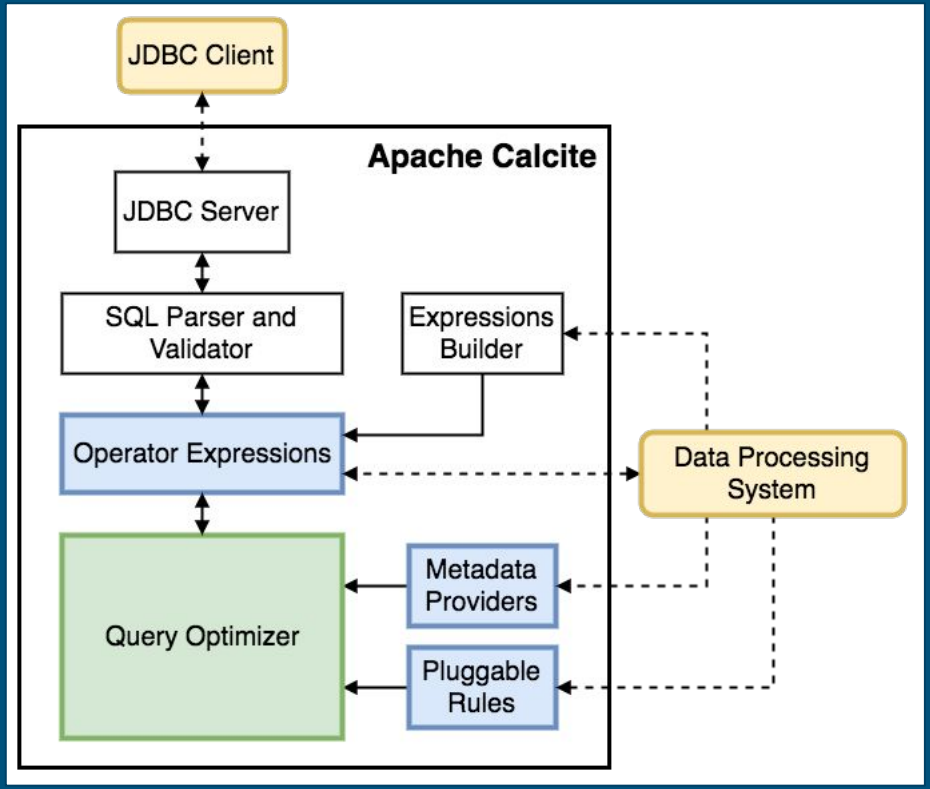
Bring database approaches to new areas (e.g. streaming)

Allow create a DBMS by composing pieces (federation, etc.)

Customize by plugging into framework, evolving framework when necessary

Apache license & governance

Architecture



Core – Operator expressions (relational algebra) and planner (based on Volcano/Cascades[2])

External – Data storage, algorithms and catalog

Optional – SQL parser, JDBC & ODBC drivers

Extensible – Planner rewrite rules, statistics, cost model, algebra, UDFs

Relational algebra

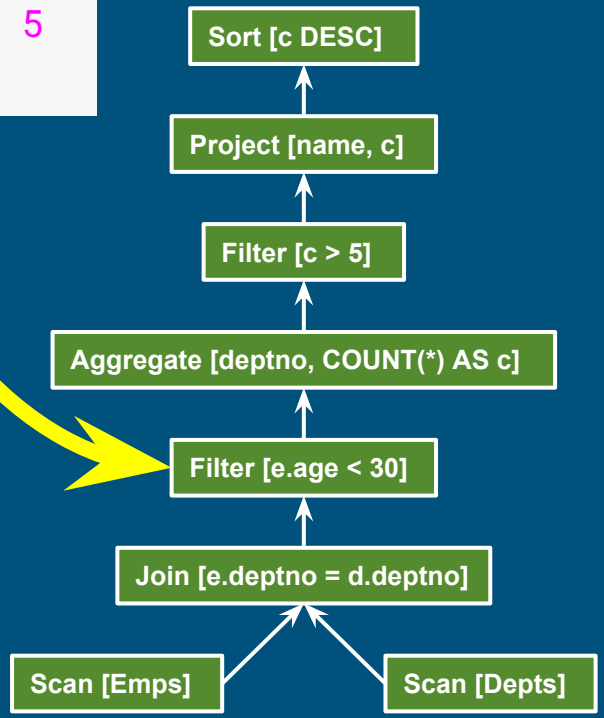
Based on set theory, plus operators: Project, Filter, Aggregate, Union, Join, Sort

Requires: declarative language (SQL), query planner

Original goal: data independence

Enables: query optimization, new algorithms and data structures

```
SELECT d.name, COUNT(*) AS c
FROM Emps AS e
JOIN Depts AS d USING (deptno)
WHERE e.age < 40
GROUP BY d.deptno
HAVING COUNT(*) > 5
ORDER BY c DESC
```



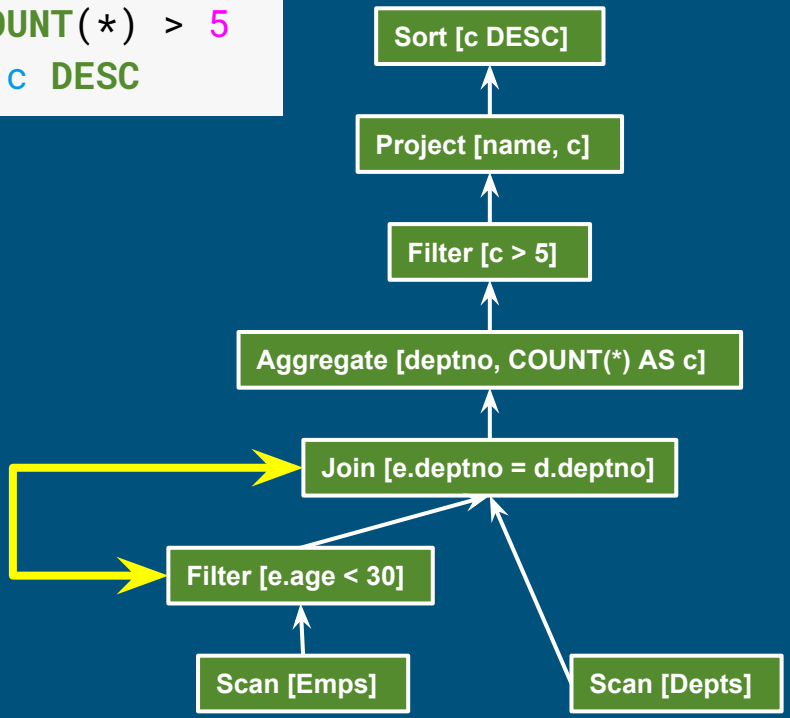
Algebraic rewrite

Optimize by applying rewrite rules that preserve semantics

Hopefully the result is less expensive; but it's OK if it's not (planner keeps "before" and "after")

Planner uses dynamic programming, seeking the lowest total cost

```
SELECT d.name, COUNT(*) AS c
FROM (SELECT * FROM Emps
      WHERE e.age < 40) AS e
JOIN Depts AS d USING (deptno)
GROUP BY d.deptno
HAVING COUNT(*) > 5
ORDER BY c DESC
```



Calcite framework

Relational algebra

RelNode (operator)
- TableScan
- Filter
- Project
- Union
- Aggregate
- …

RelDataType (type)
RexNode (expression)
RelTrait (physical property)
- RelConvention (calling-convention)
- RelCollation (sortedness)
- RelDistribution (partitioning)

RelBuilder

SQL parser

SqlNode
SqlParser
SqlValidator

Metadata

Schema
Table
Function
- TableFunction
- TableMacro

Lattice

JDBC driver

Transformation rules

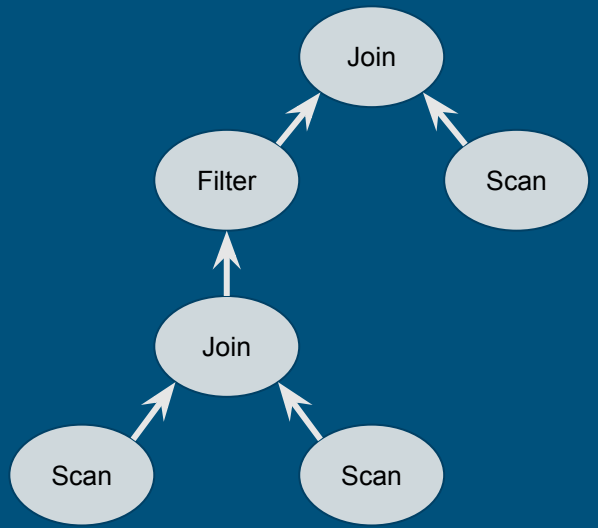
RelOptRule
- FilterMergeRule
- AggregateUnionTransposeRule
- 100+ more

Global transformations
- Unification (materialized view)
- Column trimming
- De-correlation

Cost, statistics

RelOptCost
RelOptCostFactory
RelMetadataProvider
- RelMdColumnUniquensss
- RelMdDistinctRowCount
- RelMdSelectivity

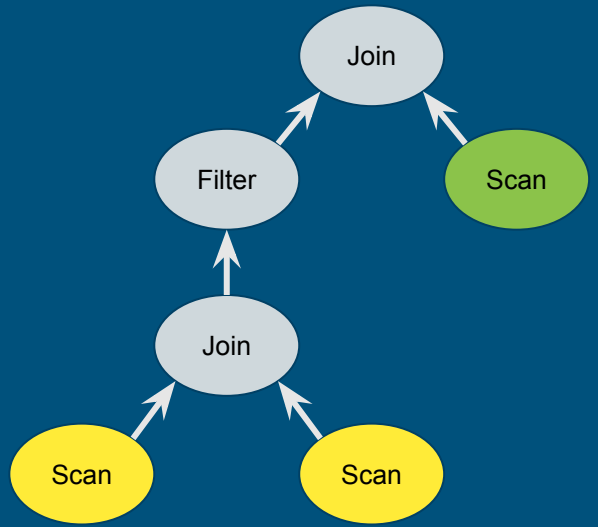
Calling convention



Initially all nodes belong to “logical” calling convention

Logical calling convention cannot be implemented, so has infinite cost

Calling convention

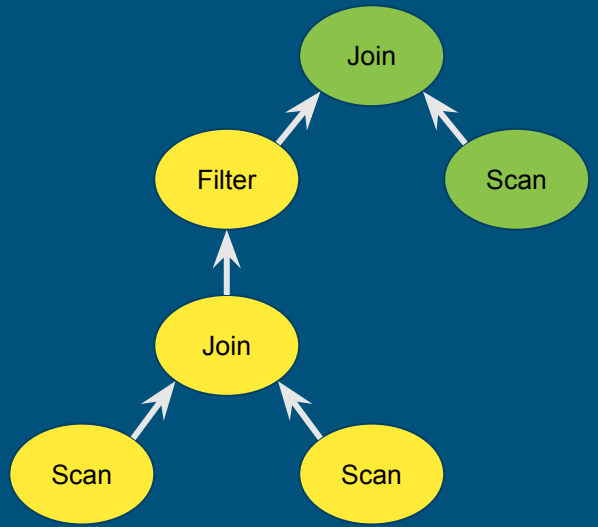


Tables can’t be moved so there is only one choice of calling convention for each table

Examples:

- Enumerable
- Druid
- Drill
- HBase
- JDBC

Calling convention



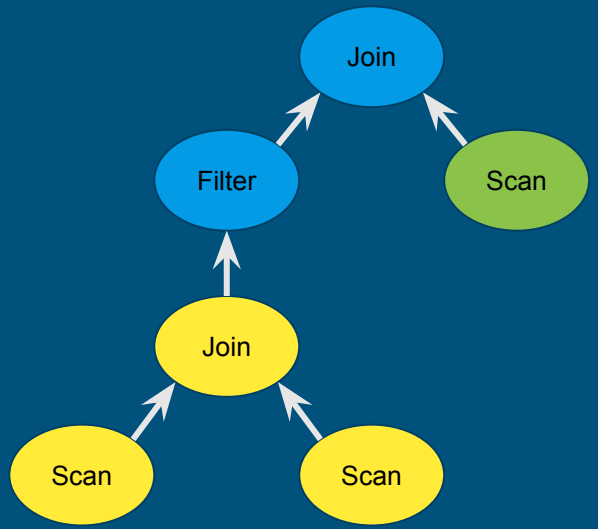
Rules fire to convert nodes to particular calling conventions

The calling convention propagates through the tree

LogicalFilter(YellowJoin) → YellowFilter(YellowJoin)

Because this is Volcano, each node can have multiple conventions

Calling convention



We also consider “engines” -- calling conventions that do not have a storage format.

Examples are Drill, Spark, Presto.

To implement, we generate program that calls out to query1 and query2.

Adapter

Implement `SchemaFactory` interface

Connect to a data source using parameters

Extract schema - return a list of tables

Push down processing to the data source:

- A set of planner rules
- Calling convention (optional)
- Query model & query generator (optional)

```
"schemas": [
 {
   "name": "HR",
   "type": "custom",
   "factory":
"org.apache.calcite.adapter.file.FileSchemaFactory",
   "operand": {
     "directory": "hr-csv"
   }
 }
]
```

```
$ ls -l hr-csv
-rw-r--r--  1 jhyde  staff   62 Mar 29 12:57 DEPTS.csv
-rw-r--r--  1 jhyde  staff  262 Mar 29 12:57 EMPS.csv.gz
$ ./sqlline  -u jdbc:calcite:model=hr.json -n scott -p tiger
sqlline> select count(*) as c from emp;
'C'
'5'
1 row selected (0.135 seconds)
```

Algebra builder

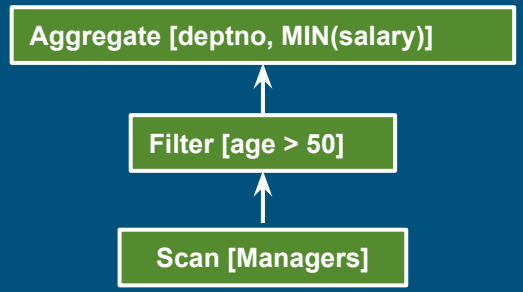
```
final FrameworkConfig config;
final RelBuilder builder = RelBuilder.create(config);
final RelNode node = builder
    .scan("EMP")
    .aggregate(
        builder.groupKey("DEPTNO"), builder.count(false, "C"),
        builder.sum(false, "S", builder.field("SAL")))
    .filter(
        builder.call(SqlStdOperatorTable.GREATER_THAN,
            builder.field("C"), builder.literal(10)))
    .build();
System.out.println(RelOptUtil.toString(node));
```

```
SELECT deptno,
  COUNT(*) AS c,
  SUM(sal) AS s
FROM Emps
HAVING COUNT(*) > 10
```

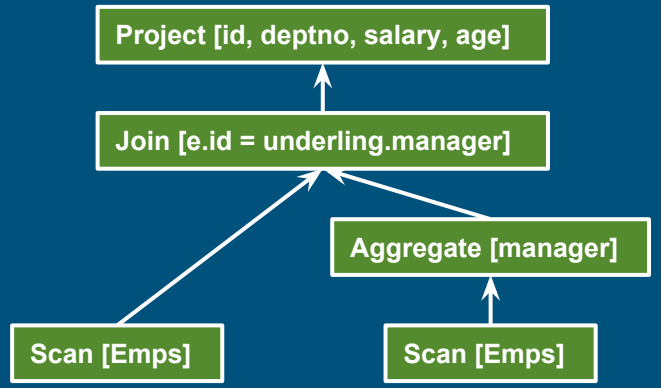
```
LogicalFilter(condition=[>($1, 10)])
    LogicalAggregate(group=[{7}], C=[COUNT()], S=[SUM($5)])
        LogicalTableScan(table=[[scott, EMP]])
```

Views

```
SELECT deptno, MIN(salary)
FROM Managers
WHERE age > 50
GROUP BY deptno
```



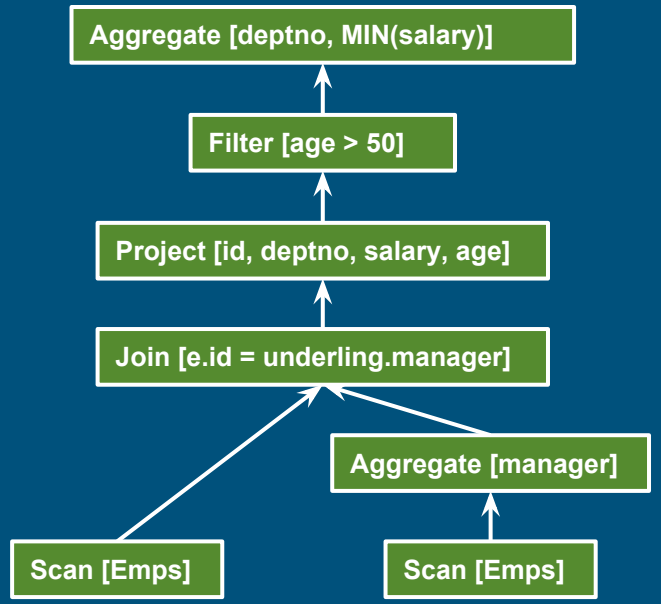

```
CREATE VIEW Managers AS
SELECT *
FROM Emps AS e
WHERE EXISTS (
  SELECT *
  FROM Emps AS underling
  WHERE underling.manager = e.id)
```



View query (after expansion)

```
SELECT deptno, MIN(salary)
FROM Managers
WHERE age > 50
GROUP BY deptno
```

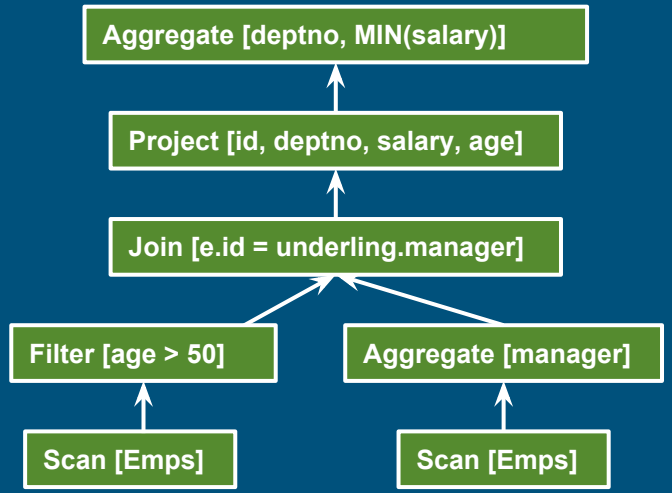
```
CREATE VIEW Managers AS
SELECT *
FROM Emps AS e
WHERE EXISTS (
  SELECT *
  FROM Emps AS underling
  WHERE underling.manager = e.id)
```



View query (after pushing down filter)

```
SELECT deptno, MIN(salary)
FROM Managers
WHERE age > 50
GROUP BY deptno
```

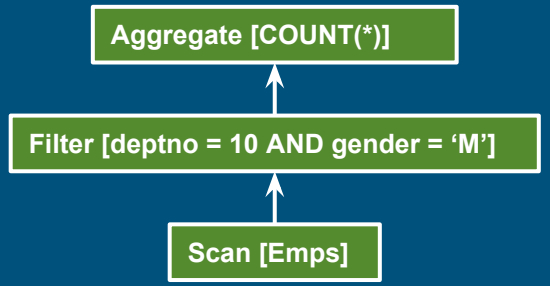
```
CREATE VIEW Managers AS
SELECT *
FROM Emps AS e
WHERE EXISTS (
  SELECT *
  FROM Emps AS underling
  WHERE underling.manager = e.id)
```



Materialized view

```
CREATE MATERIALIZED VIEW
  EmpSummary AS
SELECT deptno, gender,
  COUNT(*) AS c, SUM(sal) AS s
FROM Emps
GROUP BY deptno, gender
```

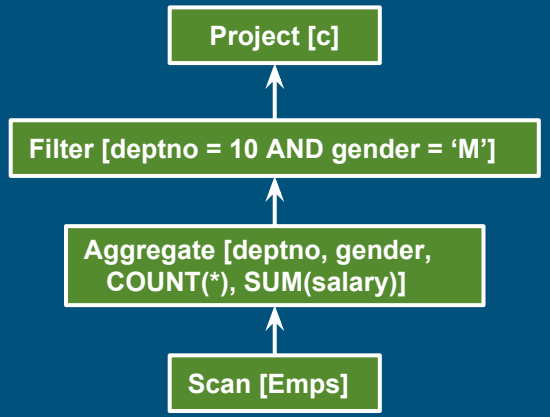

```
SELECT COUNT(*) AS c
FROM Emps
WHERE deptno = 10
AND gender = 'M'
```



Materialized view: rewrite query to match

```
CREATE MATERIALIZED VIEW
  EmpSummary AS
SELECT deptno, gender,
  COUNT(*) AS c, SUM(sal) AS s
FROM Emps
GROUP BY deptno, gender
```

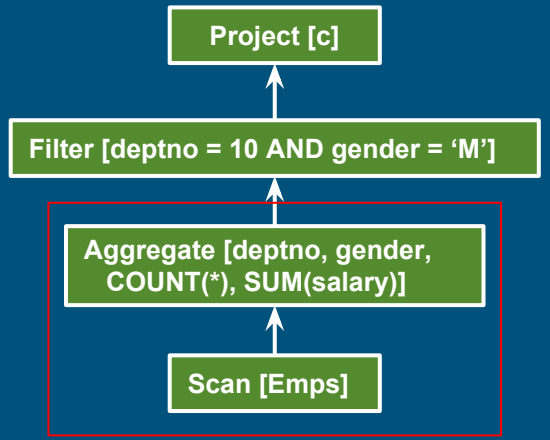

```
SELECT COUNT(*) AS c
FROM Emps
WHERE deptno = 10
AND gender = 'M'
```



Materialized view: rewrite query to match

```
CREATE MATERIALIZED VIEW
  EmpSummary AS
SELECT deptno, gender,
  COUNT(*) AS c, SUM(sal) AS s
FROM Emps
GROUP BY deptno, gender
```

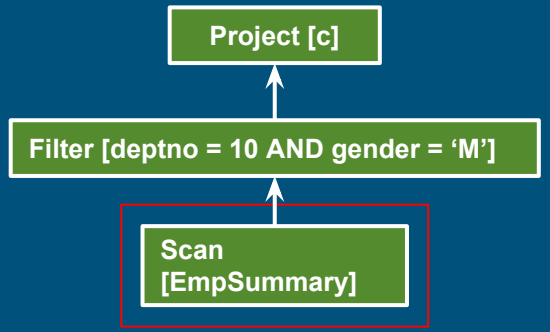

```
SELECT COUNT(*) AS c
FROM Emps
WHERE deptno = 10
AND gender = 'M'
```



Materialized view: substitute table scan

```
CREATE MATERIALIZED VIEW
  EmpSummary AS
SELECT deptno, gender,
  COUNT(*) AS c, SUM(sal) AS s
FROM Emps
GROUP BY deptno, gender
```

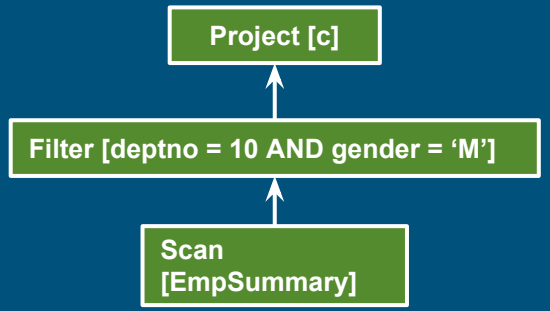

```
SELECT COUNT(*) AS c
FROM Emps
WHERE deptno = 10
AND gender = 'M'
```



Materialized view: substitute table scan

```
CREATE MATERIALIZED VIEW
  EmpSummary AS
SELECT deptno, gender,
  COUNT(*) AS c, SUM(sal) AS s
FROM Emps
GROUP BY deptno, gender
```


```
SELECT c
FROM EmpSummary
WHERE deptno = 10
AND gender = 'M'
```



Summary

Apache Calcite is a toolkit for **constructing your own DBMS**

Core is **Relational Algebra**

Optimize queries by applying **transformation rules** and choosing the best based upon **statistics**

Calling conventions allow you to represent hybrid queries

Materialized views allow you optimize your data by providing sorted, aggregate copies

Thank you!
Questions?

https://calcite.apache.org
@ApacheCalcite
@julianhyde