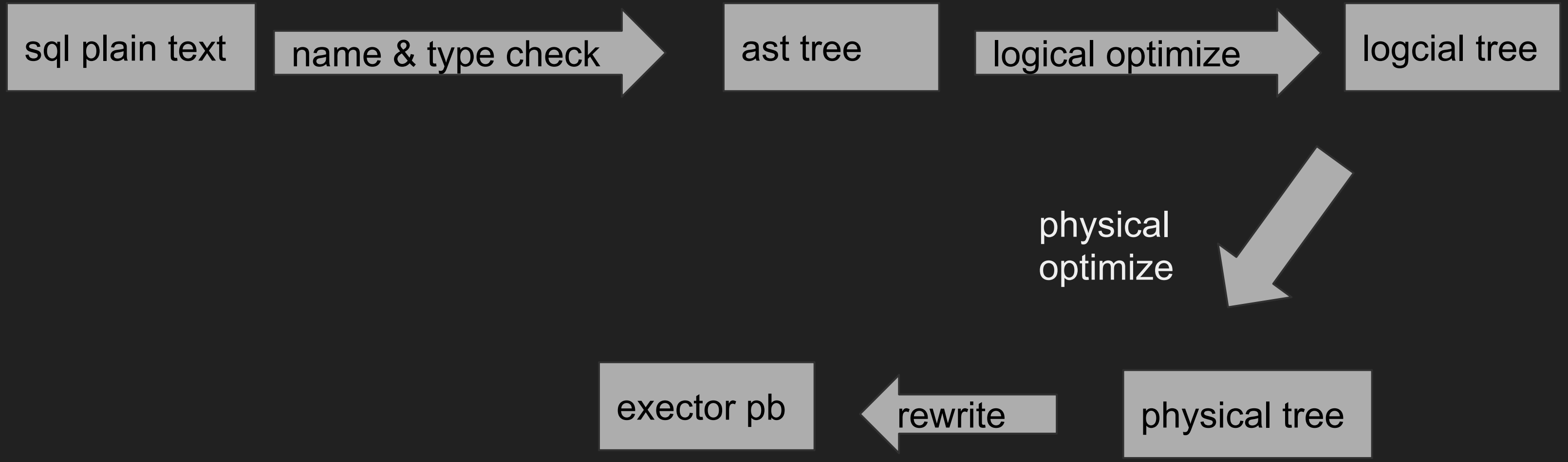
An Overview of Query Optimization

Han Fei@PingCAP

A optimizer paradigm

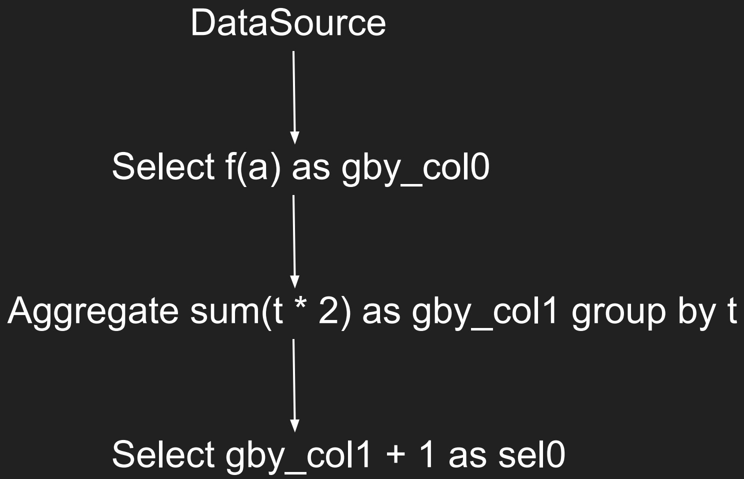


logical optimize

- common expression elimination
- column prunning
- constant fold
- constant propagation
- expression minimize
- predicate/projection/aggregation pushdown

Common Expression Elimination

- Example: Select sum(f(a) * 2) + 1 from src group by f(a)
- Can be converted to:


DataSource
Select f(a) as gby_col0
Aggregate sum(t * 2) as gby_col1 group by t
Select gby_col1 + 1 as sel0

How to identify common expression ?

Calculate Hash Code

Predicate Pushdown

- convert DNF to CNF
    - Example: A JOIN B WHERE A.id > 10 or (A.id < 5 and b.id < 5)
- split CNF items
- when predicate meets projection, aggregation, join, union, etc.
- the opt can do during ppd:
    - outer join elimination
        - Example: A LEFT OUTER JOIN B WHERE B.id < 5
    - constant propagation
        - Example: A JOIN B ON A.id = B.id WHERE A.id < 5
        - Example: A JOIN B ON A.id = B.id JOIN C ON B.id = C.id WHERE A.id < 5

physical algebra

- stream aggr & hash aggr
- index join, merge join, hash join
- subqueries, semi join
  - select * from t where t.id in (select id from s)
  - select * from t where t.id in (select count(*) from s group by s.key)
  - select * from t where t.id in (select count(*) from s group by s.key having s.id = t.key)
  - select * from t where t.id in (select count(*) from s group by s.key where s.id = t.key)
  - select count(*) from t group by t.id in (select count(*) from s group by s.key having s.id = t.key)

aggregation/distinct push down

- aggregate meets join
  - SELECT SUM(DISTINCT t.id) FROM t JOIN s ON t.id = s.id
  - SELECT SUM(DISTINCT t.id) FROM t JOIN s ON t.key = s.key
- aggregate meets union all
  - SELECT SUM(tmp.id) FROM (select * FROM T) UNION ALL (SELCT * FROM S)
  - SELECT AVG(tmp.id) FROM (select * FROM T) UNION ALL (SELCT * FROM S)

Physical Optimize

- Cost Based Optimization
- History Based Optimization
- Data Skew Optimization

Cost Modal

- Combines components of estimated
  - CPU (instructions)
  - I/O (random and sequential)
  - Communications between nodes
- Basic Statistics
  - Number of rows in tables
  - for each columns
    - distinct value, avg length of data values, data range infos (histogram)

Interesting Order, Group

- order
  - SELECT * FROM A JOIN B ON A.ID = B.ID ORDER BY A.KEY
- group
  - SELECT SUM(A.KEY) FROM A JOIN B ON A.ID = B.ID GROUP BY A.ID
  - A , B Sort by ID and apply merge join
  - A Group by ID and apply hash join

Join ReOrder

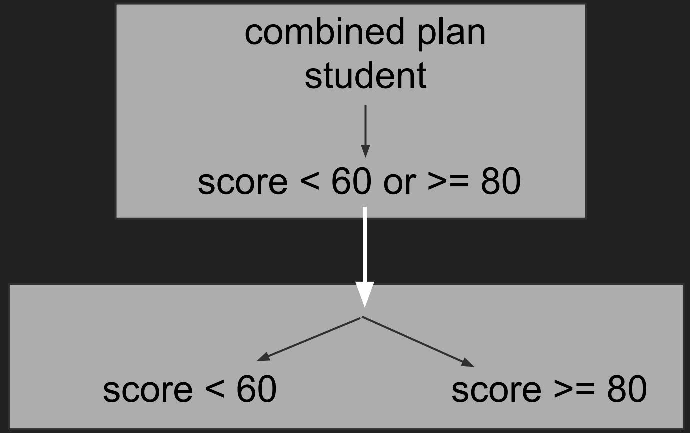
- Specify the EquivalenceClass
  - a join b on a.id = b.id join c on a.key = c.key join d on a.id = d.id
  - a b d can use merge join in one task
- For a few join items : Dynamic Programing
- Else apply a greedy algo

Adviced Topic: History based optimization

- statistics feedback and self-adapt

Adviced Topic: Common queries(tables) combine

- SELECT id from student where score < 60
- SELECT id from student where score >= 80
- Two Choices:



Skew Aggregation, Join

Search Engine

- Buttom-Up (R system -> Starburst, DB2, Oracle)
- TopDown (Volcano -> Cascades, SQL-Server, SCOPE, Calcite)

Thanks!