

# Query Optimization Lab

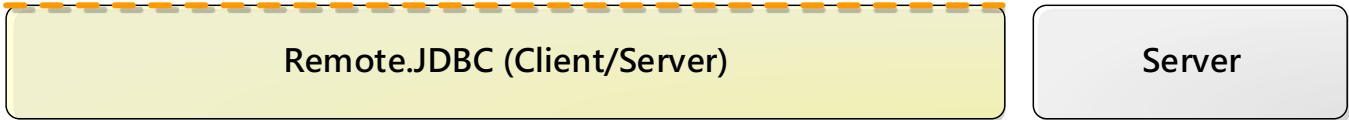
Introduction to Database Systems

2023 Spring

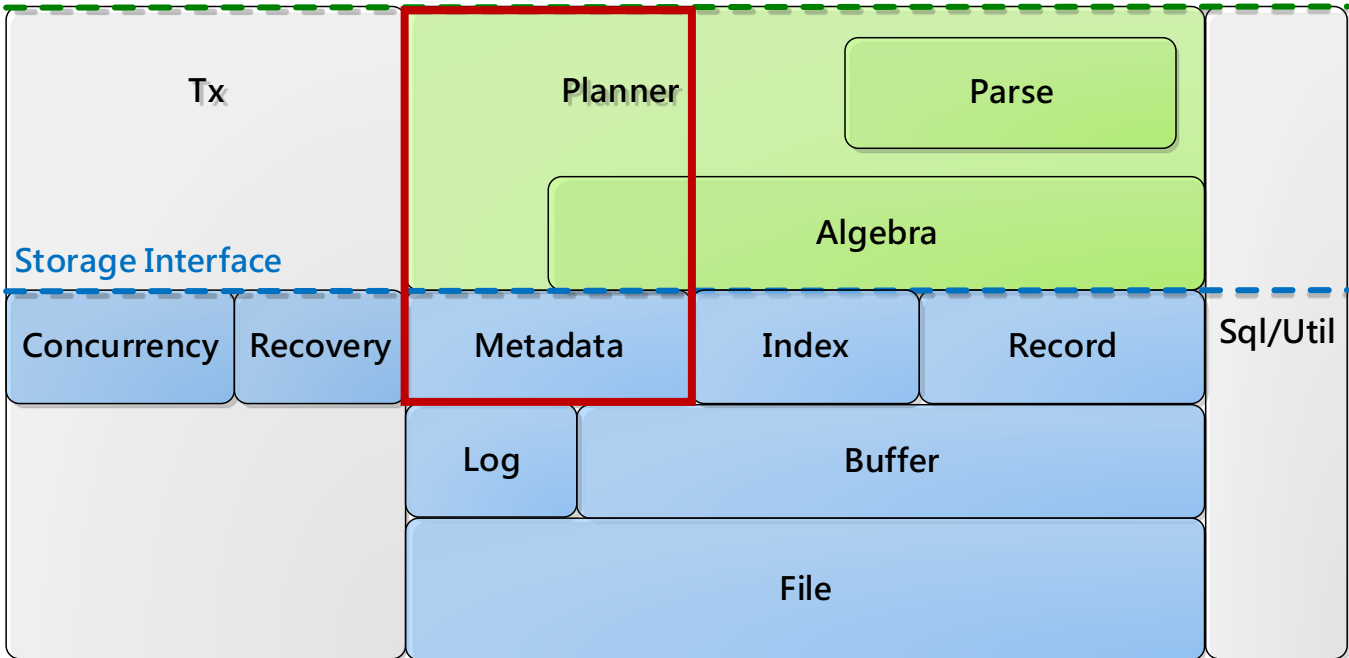
# Where Are We?

VanillaCore

JDBC Interface (at Client Side)



Query Interface



# Selinger-Style Optimizer

- In assignment 3, we used BasicQueryPlanner where table join order is fixed
- We can use SelingerLikeQueryPlanner which uses *dynamic programming* to compute a better table join order

# Query Planner in VanillaDB

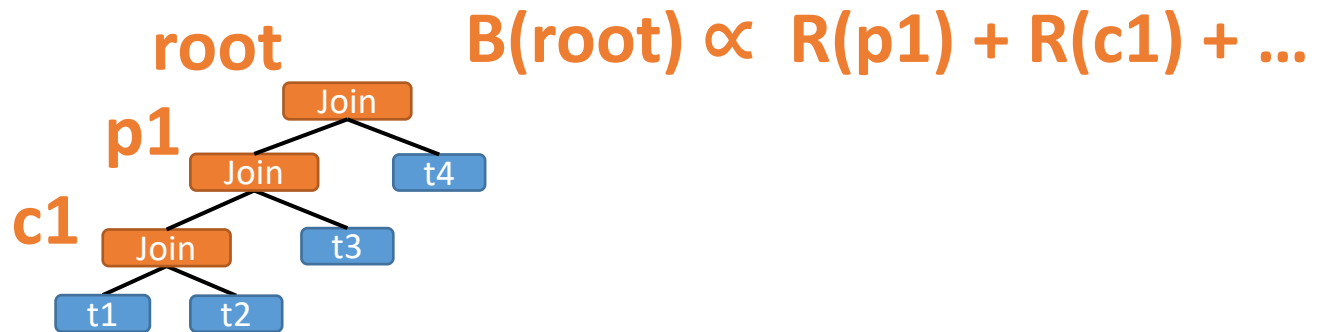
- BasicQueryPlanner
  - Produce unoptimized plan trees
- HeuristicQueryPlanner
  - Optimize queries using *predicate pushdown*
- SelingerLikeQueryPlanner
  - Optimize queries using *predicate pushdown* and *selecting a table join order that minimizes the cost*

# BasicQueryPlanner

```
public Plan createPlan(QueryData data, Transaction tx) {  
    // Step 1: Create a plan for each mentioned table or view  
    List<Plan> plans = new ArrayList<Plan>();  
    for (String tblname : data.tables()) {  
        String viewdef = VanillaDb.catalogMgr().getViewDef(tblname, tx);  
        if (viewdef != null)  
            plans.add(VanillaDb.newPlanner().createQueryPlan(viewdef, tx));  
        else  
            plans.add(new TablePlan(tblname, tx));  
    }  
    // Step 2: Create the product of all table plans  
    Plan p = plans.remove(0)  
    for (Plan nextplan : plans)  
        p = new ProductPlan(p, nextplan);  
    // Step 3: Add a selection plan for the predicate  
    p = new SelectPlan(p, data.pred());  
    // Step 4: Add a group-by plan if specified  
    if (data.groupFields() != null) {  
        p = new GroupByPlan(p, data.groupFields(), data.aggregationFn(), tx);  
    }  
    // Step 5: Project onto the specified fields  
    p = new ProjectPlan(p, data.projectFields());  
    // Step 6: Add a sort plan if specified  
    if (data.sortFields() != null)  
        p = new SortPlan(p, data.sortFields(), data.sortDirections(), tx);  
    // Step 7: Add a explain plan if the query is explain statement  
    if (data.isExplain())  
        p = new ExplainPlan(p);  
    return p;  
}
```

- Product/join order follows what's written in SQL

# Selinger-Style Optimizer



- Consider the best trees after joining 3 tables
- Observation:
  - $B(\mathbf{t3} \bowtie \mathbf{t1} \bowtie \mathbf{t2}) \leq B(\mathbf{t2} \bowtie \mathbf{t3} \bowtie \mathbf{t1})$
  - $B(\mathbf{t3} \bowtie \mathbf{t1} \bowtie \mathbf{t2} \bowtie \mathbf{t4}) \leq B(\mathbf{t2} \bowtie \mathbf{t3} \bowtie \mathbf{t1} \bowtie \mathbf{t4})$
- We can use **dynamic programming** to avoid repeating computations

# query/planner/opt/SelingerQueryPlanner

```
public Plan createPlan(QueryData data, Transaction tx) {  
    // Step 1: Create a TablePlanner object for each mentioned table/view  
    int id = 0;  
    for (String tbl : data.tables()) {  
        String viewdef = VanillaDb.catalogMgr().getViewDef(tbl, tx);  
        if (viewdef != null)  
            views.add(VanillaDb.newPlanner().createQueryPlan(viewdef, tx));  
        else {  
            TablePlanner tp = new TablePlanner(tbl, data.pred(), tx, id);  
            tablePlanners.add(tp);  
            id += 1;  
        }  
    }  
    // step 2: Use Selinger optimization to find join access path  
    Plan trunk = getAccessPath();  
    // Step 3: Add a group by plan if specified  
    if (data.groupFields() != null)  
        trunk = new GroupByPlan(trunk, data.groupFields(),  
                                data.aggregationFn(), tx);  
    // Step 4. Project on the field names  
    trunk = new ProjectPlan(trunk, data.projectFields());  
    // Step 5: Add a sort plan if specified  
    if (data.sortFields() != null)  
        trunk = new SortPlan(trunk, data.sortFields(),  
                              data.sortDirections(), tx);  
    // Step 6: Add a explain plan if the query is explain statement  
    if (data.isExplain())  
        trunk = new ExplainPlan(trunk);  
    return trunk;  
}
```

# Selinger Optimizer Example (1/6)

- Here are 3 relations to join: A, B, C
- Step 1: compute the cost (R) of each relation's cheapest plan

1-Set	Best Plan	R
{A}	Index Select Plan	10
{B}	Table Plan	30
{C}	Select Plan	20



# query/planner/opt/SelingerQueryPlanner

```
private Plan getAccessPath() {
    Plan viewTrunk = null;

    // deal with view first
    while (!views.isEmpty())
        viewTrunk = getLowestView();

    // use DP and left deep to find cheapest plan
    return getAllCombination(viewTrunk);
}
```

```
private Plan getAllCombination(Plan viewTrunk) {
    int finalKey = 0;

    // construct all combination layer by layer
    for (int layer = 1; layer <= tablePlanners.size(); layer++) {
        // when layer = 1, use select down strategy to construct layer 1
        if (layer == 1) {
            for (TablePlanner tp: tablePlanners) {
                Plan bestPlan = null;
                if (viewTrunk != null) {
                    bestPlan = tp.makeJoinPlan(viewTrunk);
                    if (bestPlan == null)
                        bestPlan = tp.makeProductPlan(viewTrunk);
                }
                else
                    bestPlan = tp.makeSelectPlan();

                AccessPath ap = new AccessPath(tp, bestPlan);
                lookupTbl.put(ap.hashCode(), ap);

                // compute final hash key
                finalKey += tp.hashCode();
            }
            continue;
        }
    }
}
```

- Select the best way to access each relation

# Selinger Optimizer Example (2/6)

- Step 2: compute the cost of 2-way joins reusing the costs from step 1

1-Set	Best Plan	Cost
{A}	Index Select Plan	10
{B}	Table Plan	30
{C}	Select Plan	20

2-Set	Best Plan	Cost
{A,B}	$A \bowtie B$	159
{A,C}	$A \bowtie C$	98
{B,C}	$B \bowtie C$	77

# Selinger Optimizer Example (3/6)

- Here are 3 relations to join – A, B, C
- Step 2
  - Compute the cost of 2-way join by estimating the costs of all permutations using the results from step 1
  - Ex.  $\{A, B\} =$ 
    - $B(\{A\} \bowtie B)$  Cost: 159 ★
    - $B(\{B\} \bowtie A)$  Cost: 189

Sub plan	Best Plan	Cost
{A}	Index Select Plan	10
{B}	Table Plan	30
{C}	Select Plan	20

Sub plan	Best Plan	Cost
{A, B}	$A \bowtie B$	159

# Selinger Optimizer Example (4/6)

- Here are 3 relations to join – A, B, C
- Step 2
  - Compute the cost of 2-way join by estimating all permutation using the single relation cost just cached
  - Ex. {A, B} =
    - $B(\{A\} \bowtie B)$  Cost: 159
    - $B(\{B\} \bowtie A)$  Cost: 189

Sub plan	Best Plan	Cost
{A}	Index Select Plan	10
{B}	Table Plan	30
{C}	Select Plan	20

Sub plan	Best Plan	Cost
{A, B}	$A \bowtie B$	159
{A, C}	$C \bowtie A$	98
{B, C}	$C \bowtie B$	77

# query/planner/opt/SelingerQueryPlanner getAllCombination

```
// when layer >= 2, iterate all existing (layer-1) combination to join with all table planners to construct next layer
for (Integer key: keySet) {
    AccessPath leftTrunk = lookupTbl.get(key);

    // go left deep
    for (TablePlanner rightOne: tablePlanners) {
        // only consider (layer-1) combination
        if (leftTrunk.getTblUsed().size() < layer-1)
            continue;

        // cannot join with table which combination already included
        if (leftTrunk.isUsed(rightOne.getId()))
            continue;

        // do join
        Plan bestPlan = rightOne.makeJoinPlan(leftTrunk.getPlan());
        if (bestPlan == null)
            bestPlan = rightOne.makeProductPlan(leftTrunk.getPlan());

        int newKey = leftTrunk.hashCode() + rightOne.hashCode();
        AccessPath ap = lookupTbl.get(newKey);

        // there is no access path contains this combination
        if (ap == null) {
            AccessPath newAp = new AccessPath(leftTrunk, rightOne, bestPlan);
            lookupTbl.put(newKey, newAp);
        }
        else {
            // check whether new access path is better than previous
            if (bestPlan.recordsOutput() < ap.getCost()) {
                AccessPath newAp = new AccessPath(leftTrunk, rightOne, bestPlan);
                lookupTbl.put(newKey, newAp);
            }
        }
    }
}
}
```

- Iterate all table planners to join with all existing (layer-1) combination to construct this layer

# query/planner/opt/SelingerQueryPlanner

```
// when layer >= 2, iterate all existing (layer-1) combination to join with all table planners to construct next layer
for (Integer key: keySet) {
    AccessPath leftTrunk = lookupTbl.get(key);

    // go left deep
    for (TablePlanner rightOne: tablePlanners) {
        // only consider (layer-1) combination
        if (leftTrunk.getTblUsed().size() < layer-1)
            continue;

        // cannot join with table which combination already included
        if (leftTrunk.isUsed(rightOne.getId()))
            continue;

        // do join
        Plan bestPlan = rightOne.makeJoinPlan(leftTrunk.getPlan());
        if (bestPlan == null)
            bestPlan = rightOne.makeProductPlan(leftTrunk.getPlan());

        int newKey = leftTrunk.hashCode() + rightOne.hashCode();
        AccessPath ap = lookupTbl.get(newKey);

        // there is no access path contains this combination
        if (ap == null) {
            AccessPath newAp = new AccessPath(leftTrunk, rightOne, bestPlan);
            lookupTbl.put(newKey, newAp);
        }
        else {
            // check whether new access path is better than previous
            if (bestPlan.recordsOutput() < ap.getCost()) {
                AccessPath newAp = new AccessPath(leftTrunk, rightOne, bestPlan);
                lookupTbl.put(newKey, newAp);
            }
        }
    }
}
}
```

- Iterate all table planners to join with all existing (layer-1) combination to construct this layer

# query/planner/opt/SelingerQueryPlanner

```
// when layer >= 2, iterate all existing (layer-1) combination to join with all table planners to construct next layer
for (Integer key: keySet) {
    AccessPath leftTrunk = lookupTbl.get(key);

    // go left deep
    for (TablePlanner rightOne: tablePlanners) {
        // only consider (layer-1) combination
        if (leftTrunk.getTblUsed().size() < layer-1)
            continue;

        // cannot join with table which combination already included
        if (leftTrunk.isUsed(rightOne.getId()))
            continue;

        // do join
        Plan bestPlan = rightOne.makeJoinPlan(leftTrunk.getPlan());
        if (bestPlan == null)
            bestPlan = rightOne.makeProductPlan(leftTrunk.getPlan());

        int newKey = leftTrunk.hashCode() + rightOne.hashCode();
        AccessPath ap = lookupTbl.get(newKey);

        // there is no access path contains this combination
        if (ap == null) {
            AccessPath newAp = new AccessPath(leftTrunk, rightOne, bestPlan);
            lookupTbl.put(newKey, newAp);
        }
        else {
            // check whether new access path is better than previous
            if (bestPlan.recordsOutput() < ap.getCost()) {
                AccessPath newAp = new AccessPath(leftTrunk, rightOne, bestPlan);
                lookupTbl.put(newKey, newAp);
            }
        }
    }
}
```

- Iterate all table planners to join with all existing (layer-1) combination to construct this layer
- Choose the better one

# Selinger Optimizer Example (5/6)

- Here are 3 relations to join – A, B, C
- Step 3
  - Compute the cost of 3-way join by estimating all left-deep tree permutation using results from step 2
  - Ex.  $\{A, B, C\} =$ 
    - $B(\{A, B\} \bowtie C)$  Cost: 259
    - $B(\{B, C\} \bowtie A)$  Cost: 111
    - $B(\{A, C\} \bowtie B)$  Cost: 100



Sub plan	Best Plan	Cost
$\{A, B\}$	$A \bowtie B$	159
$\{A, C\}$	$C \bowtie A$	98
$\{B, C\}$	$C \bowtie B$	77



# Selinger Optimizer Example (6/6)

- Here are 3 relations to join – A, B, C
- Step 3
  - Compute the cost of 3-way join by estimating all left-deep tree permutation using the step2's record
  - Ex. {A, B, C} =
    - $B(\{A, B\} \bowtie C)$  Cost: 259
    - $B(\{B, C\} \bowtie A)$  Cost: 111
    - $B(\{A, C\} \bowtie B)$  Cost: 100

Sub plan	Best Plan	Cost
{A, B}	$A \bowtie B$	159
{A, C}	$C \bowtie A$	98
{B, C}	$C \bowtie B$	77

Sub plan	Best Plan	Cost
{A, B, C}	$C \bowtie A \bowtie B$	100

# More

- An interesting work about optimizing Join Queries
- <https://arxiv.org/abs/1808.03196>