Query Optimization Lab

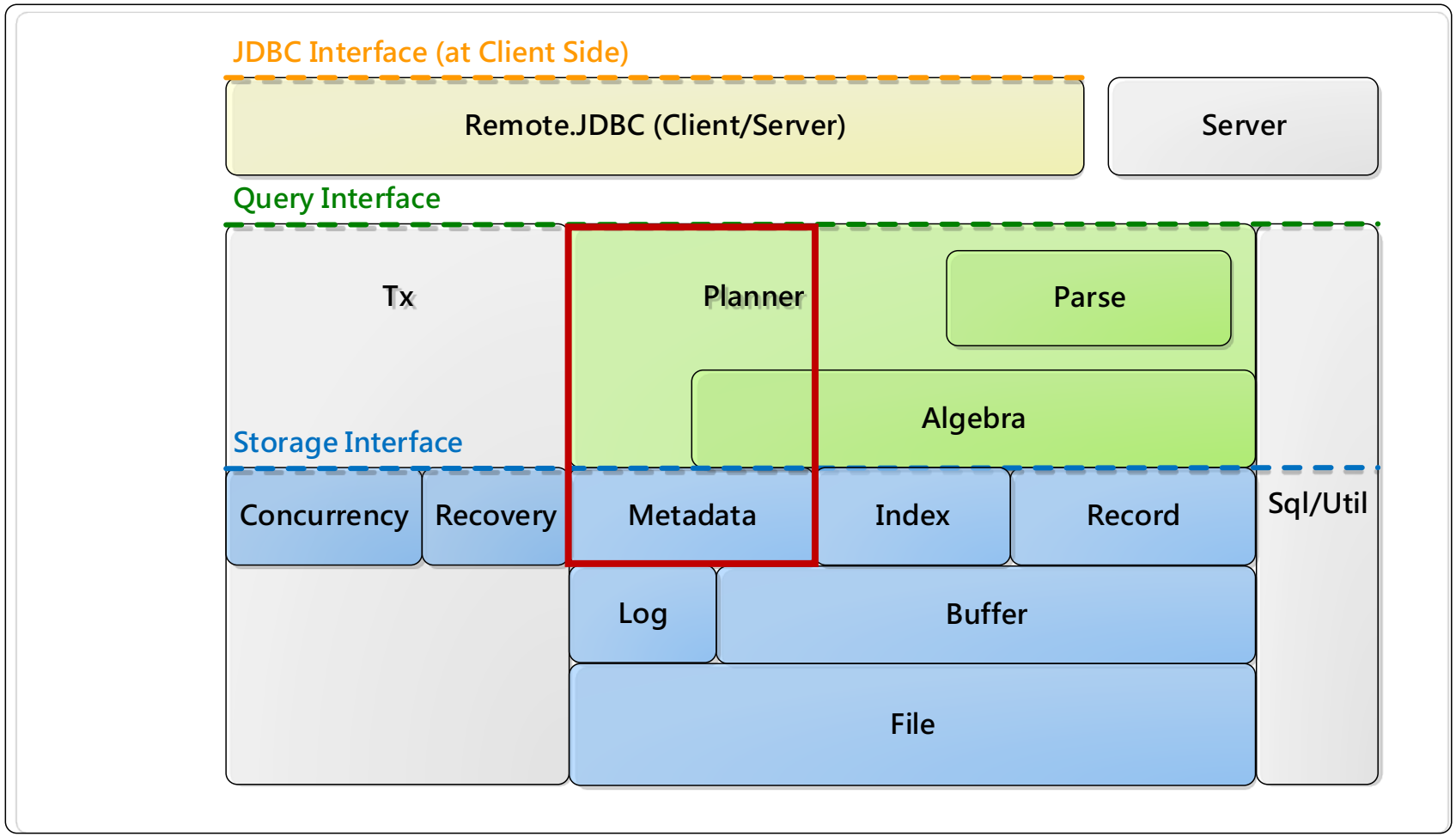
DataLab

Introduction to Database Systems

2022 Spring

Where Are We?

VanillaCore



Selinger-Style Optimizer

- In assignment3, we use BasicQueryPlanner which Product/join order is fixed.

- We have introduced another method called Selinger-Style Optimizer which use ***dynamic programming*** to compute better order of Product/join

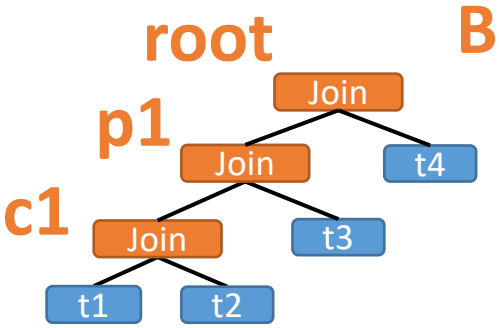
- We will trace the Selinger-Style Optimizer's code today

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



B(root) ∝ R(p1) + R(c1) + …

- Consider the best trees after 1, 2, 3, … joins
- Observation:
  - If B(**t3 ⋈ t1 ⋈ t2**) <= B(**t2 ⋈ t3 ⋈ t1**), then B(**t3 ⋈ t1 ⋈ t2** ⋈ t4) <= B(**t2 ⋈ t3 ⋈ t1** ⋈ 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;
        }
```

- bestPlan for layer 1

Selinger Optimizer Example (2/6)

- Step 2: compute the cost of 2-way joins reusing 1-way cost just cached

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⋈B	159
{A,C}	A⋈C	98
{B,C}	B⋈C	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 (3/6)

- 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} ⋈ B) Cost: 159 ★
    - B({B} ⋈ 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⋈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} ⋈ B) Cost: 159
    - B({B} ⋈ 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⋈B	159
{A, C}	C⋈A	98
{B, C}	C⋈B	77

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 the step2's record
  - Ex. {A, B, C} =
    - B({A, B} ⋈ C) Cost: 259
    - B({B, C} ⋈ A) Cost: 111
    - B({A, C} ⋈ B) Cost: 100

Sub plan	Best Plan	Cost
{A, B}	A⋈B	159
{A, C}	C⋈A	98
{B, C}	C⋈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} ⋈ C) Cost: 259
    - B({B, C} ⋈ A) Cost: 111
    - B({A, C} ⋈ B) Cost: 100

Sub plan	Best Plan	Cost
{A, B}	A⋈B	159
{A, C}	C⋈A	98
{B, C}	C⋈B	77

Sub plan	Best Plan	Cost
{A, B, C}	C⋈A⋈B	100

query/planner/opt/TablePlanner

```
public TablePlanner(String tblName, Predicate pred, Transaction tx, int id) {
    this.tblName = tblName;
    this.pred = pred;
    this.tx = tx;
    this.id = id;
    this.hashCode = (int) Math.pow(2, id);
    tp = new TablePlan(tblName, tx);
    sch = tp.schema();
}
```

query/planner/opt/AcessPath

```
public class AccessPath {
    private Plan p;
    private int hashCode = 0;
    private ArrayList<Integer> tblUsed = new ArrayList<Integer>();

    public AccessPath (TablePlanner newTp, Plan p) {
        this.p = p;
        this.tblUsed.add(newTp.getId());
        this.hashCode = newTp.hashCode();
    }

    public AccessPath (AccessPath preAp, TablePlanner newTp, Plan p) {
        this.p = p;
        this.tblUsed.addAll(preAp.getTblUsed());
        this.tblUsed.add(newTp.getId());
        this.hashCode = preAp.hashCode() + newTp.hashCode();
    }
```

- Using **pow(2, tp.id)** to avoid problems with different combinations but with the same apID
- Using **sum of pow(2, tp.id)** to represent the combination of tables in this access path
- Then we can use apID as the key of the lookup table

More

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