Query Optimization

Kyuseok Shim

KDD Laboratory
http://kdd.snu.ac.kr/
Seoul National University

1

Query Optimization

- To process an SQL query,
  - database systems must select the most efficient plan
- Very expensive
  - The number of alternative plans for a query grows at least exponentially with the number of tables
- Cost-based optimization
  - Cost estimation of operators
  - Selectivity estimation is required

2

Query Optimization

- An important task in a relational DBMS.
- Must understand optimization in order to understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).

3

Query Optimization

- Plan – A tree of relational algebra operators with choices of algorithms for each operator
- Two main issues:
  - For a given query, what plans are considered? – search space
  - How to estimate the cost of a plan?
- Practically, we want to avoid worst plans!

4

System R Optimizer

- Widely used currently – works well for < 10 joins
- Cost estimation
  - Approximation
  - Statistics are maintained in system catalogs to estimate costs of operations and result sizes.
  - Considers combination of CPU and I/O costs
- Search Space
  - Too large
  - *left-deep plans*
    - Left-deep plans allow output of each operator to be *pipelined* into the next operator without storing it in a temporary relation.
  - Cartesian products are avoided

5

Schema for Examples

Sailors (sid: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (sid: integer, bid: integer, day: dates, *rname*: string)

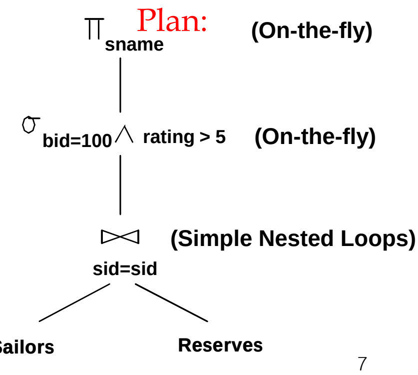
- Reserves
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

6

A Motivating Example

- Reserves – 1000 pages
  - Each tuple is 40 bytes long, 100 tuples per page
- Sailors – 500 pages
  - Each tuple is 50 bytes long, 80 tuples per page
- Simple Nested Loop Join
  - Cost: 500+500*1000 I/Os
- Misses several opportunities: selections could have been `pushed` earlier, no use is made of any available indexes, etc.

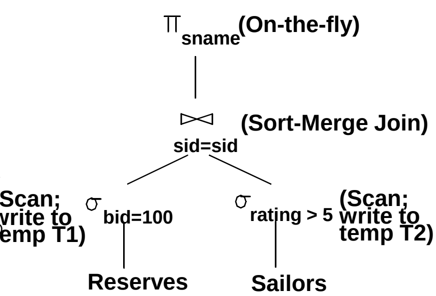
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
R.bid=100 AND S.rating>5



Alternative Plan 1

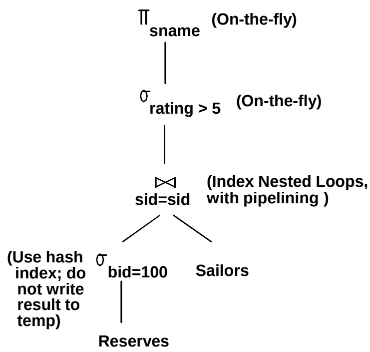
- *Push selection as early as possible*
- With 5 buffers
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if we have 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*3*250), merge (10+250)
  - Total: 3560 page I/Os.
- If we use Block Nested Loop join, join cost = 10+4*250, total cost = 2770.
- If we `push?projections, T1 has only *sid*, T2 only *sid* and *sname*:
  - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 2000.

- Reserves – 1000 pages
  - Each tuple is 40 bytes long, 100 tuples per page
- Sailors – 500 pages
  - Each tuple is 50 bytes long, 80 tuples per page



Alternative Plan 2

- With clustered index on *bid* of Reserves, we get 100,000/100 = 1000 tuples on 1000/100 = 10 pages.
- INL with *pipelining* (outer is not materialized)
  - Projecting out unnecessary fields from outer doesn't help
- Join column *sid* is a key for Sailors.
  - At most one matching tuple, unclustered index on *sid* OK.
- Decision not to push *rating>5* before the join is based on availability of *sid* index on Sailors.
- Cost: Selection of Reserves tuples (10 I/Os); for each,
- must get matching Sailors tuple (1000*1.2); total 1210 I/Os.



9

Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
  - Must estimate *size of result* for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.
- System R cost estimation approach
  - Very inexact, but works ok in practice.
  - More sophisticated techniques known now.

10

Statistics and Catalogs

- Need information about the relations and indexes involved. ***Catalogs*** typically contain at least:
  - # tuples (NTuples) and # pages (NPages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.
- Catalogs are updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.
- More detailed information (e.g., histograms of the values in some field) are sometimes stored.

11

Selectivity

SELECT attribute list
FROM relation list
WHERE cond1 AND ... AND condk

- Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- *selectivity (SF)* associated with each *condition* reflects the impact of the *condition* in reducing result size.
- *Result cardinality* = (Max # tuples) * product of all selectivities
  - Implicit assumption that *conditions* are independent!
  - Term *col=value* has SF *1/NKeys(I)*, given index I on *col*
  - Term *col1=col2* has SF *1/MAX(NKeys(I1), NKeys(I2))*
  - Term *col>value* has SF *(High(I)-value)/(High(I)-Low(I))*

12

Units of Optimization

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.age IN
    (SELECT  MAX (S2.age)
     FROM  Sailors S2
     GROUP BY  S2.rating)
```

Outer block *Nested block*

- An SQL query is parsed into a collection of *query blocks*, and these are optimized one block at a time.
- Nested blocks are usually treated as calls to a subroutine, made once per outer tuple.

13

Relational Algebra Equivalences

- Allow us to choose different join orders and to `push` selections and projections ahead of joins.
- *Selections*: $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots \sigma_{cn}(R))$ (*Cascade*)
  $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (Commute)
- *Projections:* $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{an}(R)))$ (Cascade)
- *Joins:* $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ (Associative)
  $(R \bowtie S) \equiv (S \bowtie R)$ (Commute)

+ Show that: $R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S$

14

More Equivalences

- A projection commutes with a selection that only uses attributes retained by the projection.
- Selection between attributes of the two arguments of a cross-product converts cross-product to a join.
- A selection on just attributes of R commutes with $R \bowtie S$. (i.e., $\sigma(R \bowtie S) \equiv \sigma(R) \bowtie S$)
- Similarly, if a projection follows a join $R \bowtie S$, we can `push` it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.

15

Enumeration of Alternative Plans

- There are two main cases:
  - Single-relation plans
  - Multiple-relation plans
- For queries over a single relation, queries consist of a combination of selects, projects, and aggregate ops:
  - Each available access path (file scan / index) is considered, and the one with the least estimated cost is chosen.
  - The different operations are essentially carried out together (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are *pipelined* into the aggregate computation).

16

Cost Estimates for Single-Relation Plans

- Index I on primary key matches selection:
  - *Cost is Height(I)+1 for a B+ tree, about 1.2 for hash index.*
- Clustered index I matching one or more selects:
  - *(NPages(I)+NPages(R)) * product of SFs of matching selects.*
- Non-clustered index I matching one or more selects:
  - *(NPages(I)+NTuples(R)) * product of SFs of matching selects.*
- Sequential scan of file:
  - *NPages(R).*
- *Note: Typically, no duplicate elimination on projections! (Exception: Done on answers if user says DISTINCT.)*

17

An Example

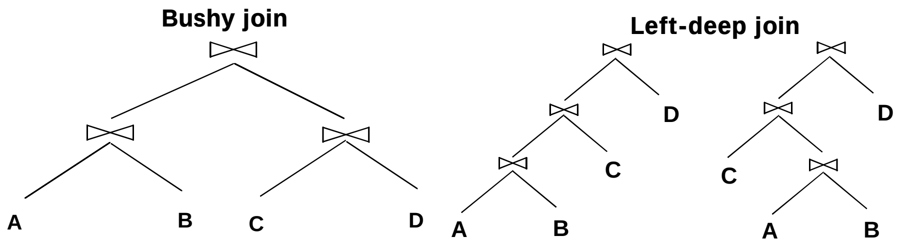
```
SELECT S.sid
FROM Sailors S
WHERE S.rating=8
```

- If we use an index on *rating*:
  - (1/NKeys(I)) * NTuples(R) = (1/10) * 40000 tuples retrieved.
  - Clustered index: (1/NKeys(I)) * (NPages(I)+NPages(R)) = (1/10) * (50+500) pages are retrieved. (This is the *cost*.)
  - Unclustered index: (1/NKeys(I)) * (NPages(I)+NTuples(R)) = (1/10) * (50+40000) pages are retrieved.
- Doing a file scan:
  - We retrieve all file pages (500).

18

Queries Over Multiple Relations

- Fundamental decision in System R: *only left-deep join trees* are considered.
  - As the number of joins increases, the number of alternative plans grows rapidly; *we need to restrict the search space.*
  - Left-deep trees allow us to generate all *fully pipelined* plans.
    - Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).



19

Enumeration of Left-Deep Plans

- Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. *(All 2-relation plans.)*
  - .....
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N뭩h relation. *(All N-relation plans.)*
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

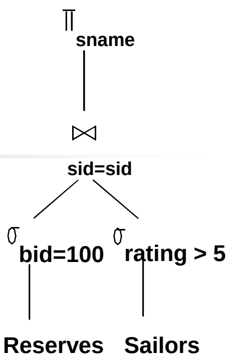
20

Enumeration of Plans (Contd.)

- ORDER BY, GROUP BY, aggregates etc. handled as a final step, using either an `interestingly ordered` plan or an additional sorting operator.
- An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
  - i.e., avoid Cartesian products if possible.
- In spite of pruning plan space, this approach is still exponential in the # of tables.

21

Example



Sailors:
B+ tree on *rating*
Hash on *sid*
Reserves:
B+ tree on *bid*

- Pass1:
  - *Sailors*: B+ tree matches *rating>5*, and is probably cheapest. However, if this selection is expected to retrieve a lot of tuples, and index is unclustered, file scan may be cheaper.
    - Still, B+ tree plan kept (because tuples are in *rating* order).
  - *Reserves*: B+ tree on *bid* matches *bid=500*; cheapest.
- Pass 2:
  - We consider each plan retained from Pass 1 as the outer, and consider how to join it with the (only) other relation.
    - e.g., *Reserves as outer*: Hash index can be used to get Sailors tuples that satisfy *sid* = outer tuple의 *sid* value.

22

Nested Queries

```
SELECT  S.sname
FROM  Sailors S
WHERE EXISTS
   (SELECT  *
    FROM  Reserves R
    WHERE  R.bid=103
     AND  R.sid=S.sid)
```

- Nested block is optimized independently, with the outer tuple considered as providing a selection condition.
- Outer block is optimized with the cost of `calling` nested block computation taken into account.
- Implicit ordering of these blocks means that some good strategies are not considered.

<u>Nested block to optimize:</u>

SELECT *
FROM Reserves R
WHERE R.bid=103
   AND S.sid= *outer value*

<u>Equivalent non-nested query:</u>

SELECT DISTINCT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
   AND R.bid=103

23

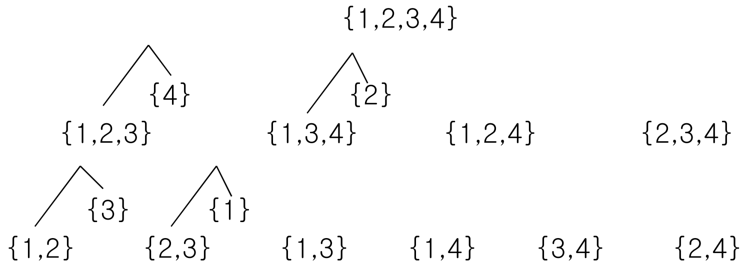
Dynamic Programming Algorithm

DP_Algorithm_LD
for i=2 to n do
  for all S⊆{R_1,⋯,R_n} such that |S| = i do
    bestPlan = a dummy plan with infinite cost
    for all R_j, S_j such that S = {R_j} ∩Sj = ø do {
      p = joinPlan(optPlan(S_j), R_j)
      if cost(p) < cost(bestPlan)
        bestPlan = p
    }
    optPlan(S) = bestPlan
  }
}
return(optPlan({R1,⋯,Rn}))

- Naïve enumeration: O(n!)
- DP Algorithm: $O(n2^{n-1})$

24

DP_Algorithm_LD



25

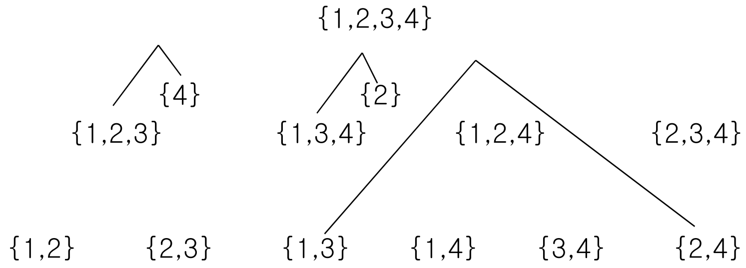
Dynamic Programming Algorithm

```
DP_Algorithm_Bushy
for i=2 to n do
    for all S⊆{R1,…,Rn} such that |S| = i do
        bestPlan = a dummy plan with infinite cost
        for all S1, S1 such that S = S1 ∪S2, S1 ≠ ø, S2 ≠ ø, S = S1 ∩S2 = ø
        do {
            p = joinPlan(optPlan(Sj), Rj)
            if cost(p) < cost(bestPlan)
                bestPlan = p
        }
        optPlan(S) = bestPlan
    }
}
return(optPlan({R1,…,Rn}))
```

- DP Algorithm: $O(3^n)$

26

DP_Algorithm_LD_Bushy



27

Query Optimization with User-define Predicates

ACM Transaction on Database Systems 24(2): 1999

28

Outline

- Motivation
  - User-defined predicates
  - Desirable execution space
- Past work
  - LDL project
  - Predicate migration
- Algorithms [VLDB 96], [ACM TODS 99]
  - An optimization algorithm that guarantees the optimal plan
  - A remarkably good approximate algorithm
- Experimental Studies

29

User-Defined Predicates

- User defined predicates (stored procedures) capture application logic
- They can be stored and executed at the server
- Can be invoked in an SQL query
- Enriches the functionality of SQL
- Raises execution and optimization challenges

30

An Example (From [Hellerstein 95])

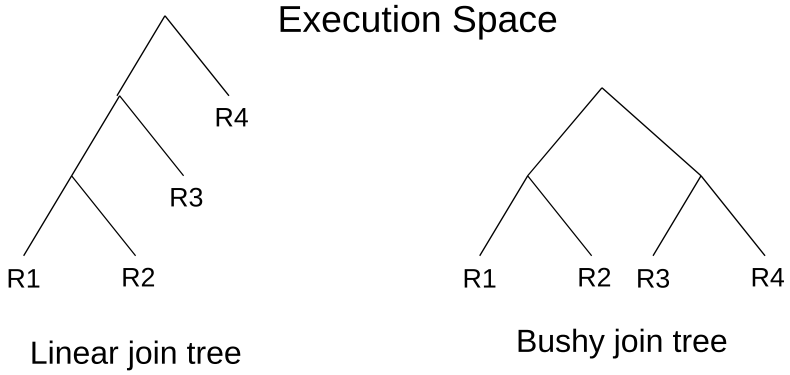
Select raster images and corresponding notes

```
select rasters.name, notes.note
from   rasters, notes
where  rasters.rtime = notes.rtime
and  rasters.rtime  < 20
and  notes.author = "clifford"
and  veg(rasters.raster) > 20
```

31

System R Style Optimization Algorithm

Execution Space



32

System R Style Optimization Algorithm

- Push down all selections
- Build plans bottom-up using DP algorithm
- Enumeration complexity – exponential in the number of relations
  - Linear join trees: $O(n2^{n-1})$
  - Bushy join trees: $O(3^n)$

{1234}

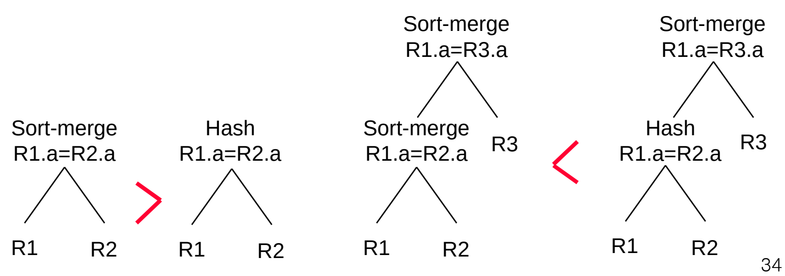
{123} {134} {124} {234}

{12} {23} {13} {14} {34} {24}

33

System R Style Optimization Algorithm

- Interesting order
  - Assume sort-merge join costs more than hash join for a join with R1 and R2
  - Sorted order resulted by a sort-merge join can reduce the cost of the extended plan from it
  - Thus, the plan with sort-merge join is additionally stored



34

User-Defined Predicates

- Evaluating user-defined predicate as early as possible is not necessarily a good idea
  - Checking whether 20% of the image have signs of vegetation takes time to evaluate
- How do we optimize queries containing user-defined predicates?
  - Cost Model
  - Execution Space

35

Cost Model

- Follow the cost models in past work
  - [Chimenti, Gamboa and Krishnamurthy 89], [Hellerstein and Stonebraker 93]
  - Selectivity, Cost per tuple
    - Cost of checking whether a raster-image has at least 20% vegetation
  - For example, Illustra allows break-up of cost into several parameters (invocation, input size,..)

36

Question

- Given
  - A single relation
  - Selection predicates f_1, f_2, …, f_n
    - their selectivities $s_1,s_2,…,s_n$
    - their costs per tuple c_1, c_2, …, c_n
- What is optimal ordering of the predicates f_1, f_2, …, f_n to process?

37

Question

- Given
  - A single relation
  - Selection predicates f_1, f_2, …, f_n
    - their selectivities $s_1,s_2,…,s_n$
    - their costs per tuple c_1, c_2, …, c_n
  - where $c_1=c_2= …=c_n$
- What is optimal ordering of the predicates f_1, f_2, …, f_n to process?

38

Question

- Given
  - A single relation
  - Selection predicates f_1, f_2, …, f_n
    - their selectivities $s_1, s_2, \dots, s_n$
    - their costs per tuple c_1, c_2, …, c_n
  - where $c_1 = c_2 = \dots = c_n$
- What is optimal ordering of the predicates f_1, f_2, …, f_n to process?
- Answer:
  - Increasing order of selectivities

39

Question

- Given
  - A single relation
  - Selection predicates f_1, f_2, …, f_n
    - their selectivities $s_1, s_2, \dots, s_n$
    - their costs per tuple c_1, c_2, …, c_n
  - where $s_1 = s_2 = \dots = s_n$
- What is optimal ordering of the predicates f_1, f_2, …, f_n to process?

40

Question

- Given
  - A single relation
  - Selection predicates f_1, f_2, …, f_n
    - their selectivities s_1,s_2,…,s_n
    - their costs per tuple c_1, c_2, …, c_n
  - where s_1=s_2= …=s_n
- What is optimal ordering of the predicates f_1, f_2, …, f_n to process?
- Answer:
  - Increasing order of costs

41

Question

- Given
  - A single relation
  - Selection predicates f_1, f_2, …, f_n
    - their selectivities s_1,s_2,…,s_n
    - their costs per tuple c_1, c_2, …, c_n
- What is the optimal ordering of the predicates f_1, f_2, …, f_n to process?

42

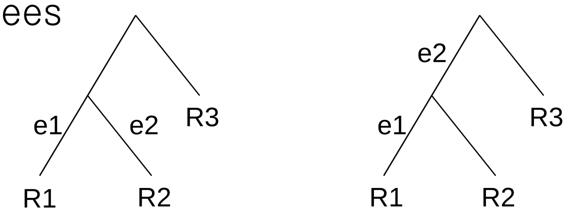
Question

- Given
  - A single relation
  - Selection predicates $f_1, f_2, \ldots, f_n$
    - their selectivities $s_1, s_2, \ldots, s_n$
    - their costs per tuple $c_1, c_2, \ldots, c_n$
- What is the optimal ordering of the predicates $f_1, f_2, \ldots, f_n$ to process?
- Answer: [Monma and Sidney 79]
  - Rank = cost/(1-selectivity)
  - Increasing order of rank

43

Desirable Execution Space

Unconstrained Linear Join Trees



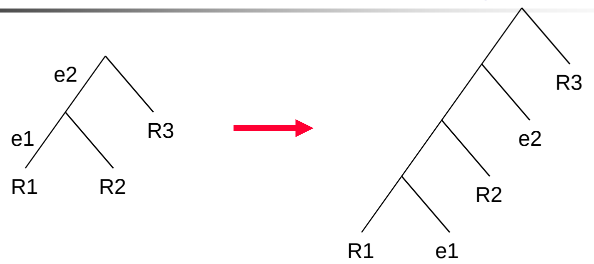
44

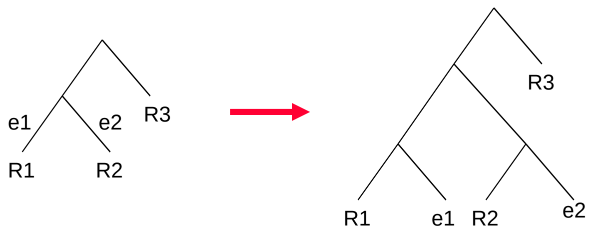
Past Work: LDL Project

- [Chimenti, Gamboa and Krishnamurthy 89]
- Treat a user-defined selection as a virtual relation with infinite cardinality
  - Plan is a linear sequence of operators
  - No dual-push-down execution plans: Join (pred1(R), pred2(S))
  - Troublesome for relatively cheap predicates
- Exponential in number of relations and user-defined predicates

45

Past Work: LDL Project





46

Past Work: Predicate Migration

- [Hellerstein and Stonebraker 93], [Hellerstein 94]
- Selections can be ordered by
  - Rank: cost/(1−selectivity)
  - Ascending order [Monma and Sidney 79]
- Treat a join predicate as a selection
  - Assume join cost is linear :
    JoinCost (R, S) = a + b*R + c* S
    c.f.) nested-loop join, user-defined join predicate
  - Assign a rank for each join predicate
- Enumerate possible join trees
- For every join tree, consider placing selections at the optimal place

47

Past Work: Predicate Migration

- Unfortunately, *fails to guarantee the optimal*
  - Needs a priori decision on which selections are evaluated before the join
  - Assume all user-defined selections are applied before the join

Sep 3, 1996 48

Past Work: Predicate Migration

- Poor integration with dynamic programming
  - Use PullRank to find an optimal plan for each join
  - If the optimal plan for join has any user-defined predicate pushed, mark unpruneable
  - Mark a subplan unpruneable if it contains unprunable subplan within it
  - Saves subplans unpruneable as well as interesting ordered
- Polynomial in number of user-defined predicates
- But can be as worse as O(n!) in the number of joins n

49

Annotating Plans with Tags

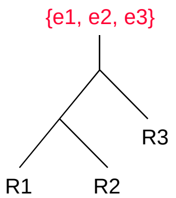
- Concept of Property
  - [Graefe and Dewitt 87],
    [Lee, Freytag and Lohman 88]
  - Two plans that represent the same expression can be compared: Join (pred1(R),S) and
    pred1(Join (R,S))
- Attach a prefix (*tag*) to every plan
  - The tag lists the set of yet to be evaluated user-defined predicates applicable to the plan
  - < e2 > {R2, R3, R4}
- We can use the traditional algorithm (almost)

Sep 3, 1996 50

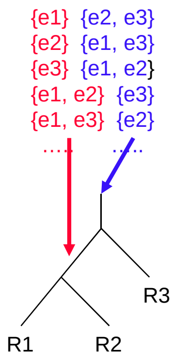
Naive Optimization Algorithm

Exponential with # of UDFs

Rank order: {e1, e2, e3}



Only one ordering by rank



All possible subset of {e1, e2, e3} => 2^k

Naive Optimization Algorithm

- Integrates well with dynamic programming algorithm
- Two plans are comparable only if both the set of relations and the tag are the same
- Very robust!
  - No assumption on the cost model
- But, exponential number of tags for every subplan

Selection Ordering

- Evaluation of a set of predicates on a relation can be ordered by
  - Rank = cost/(1−selectivity)
- Can we use ranks to reduce the number of tags?
  - Need to show that selections are ordered by rank even when separated by joins and selections
  - True if join formulas are of the form:
    JoinCost (R, S) = a + b*R + c* S + d*R*S
  - Satisfied for common join methods

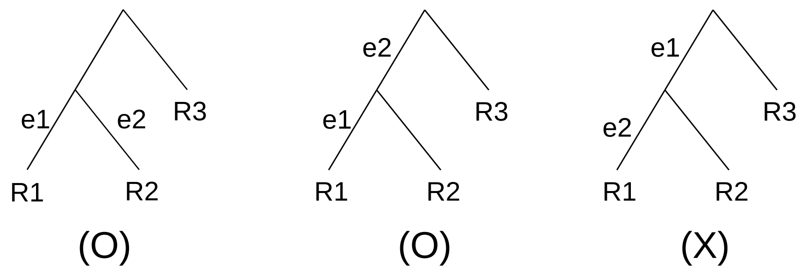
Sep 3, 1996 53

Selection Ordering

- Theorem
  - If T is any constrained execution tree using only regular join methods, then there must exist an equivalent unconstrained execution tree T\` such that cost(T\`) ≤ cost(T) and the user-defined predicates in T\` are rank-ordered.
  - Proof: See [ACM TODS 99]

54

Exploiting Rank Ordering



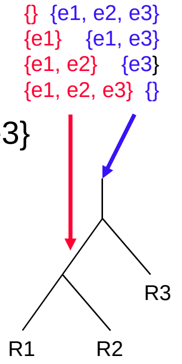
Optimization Algorithm With Complete Rank-Ordering

- At the time of every join, we consider evaluating all remaining evaluable predicates prior to join.
  - If a predicate with a rank j is applied, then so must all predicates with rank less than j.
  - We need at most $(1+w)^u$ tags
    - w: max(number of user-defined selections, number of user-defined join predicates)
    - u: sum of number of user-defined selections and number of pairs of relations having user-defined join predicates

Exploiting Rank Ordering

Polynomial with # of UDFs

Rank order: {e1, e2, e3}



2^k => 1+k

Optimization Algorithm with Complete Rank-Ordering

- Consider the step of constructing the optimal plan for the join between an intermediate relation S and a base relation R

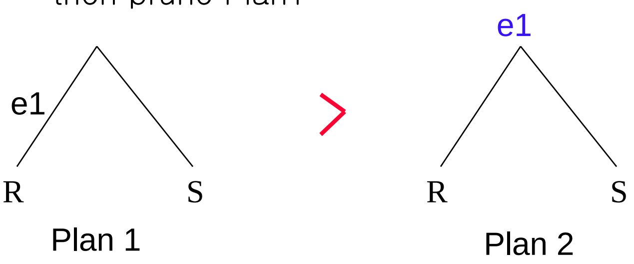
```
……………………………………………
for all u := 0  to s do
  for all v := 0  to r do
    p := extjoinPlan(optPlan(S), R, u, v)
    if  addtotable(p) then
        remove pruneset(p)
    add p to Plantable
```

Pruning Strategies

- Compare and prune plans with different tags
- UDP Push Down Rule
  - If cost(Plan 1) > cost(Plan 2) then prune Plan 1

Pruning Strategies

- UDP Pullover Rule
  - If cost(Plan 1) > cost(Plan2)+evaluation of e1, then prune Plan1

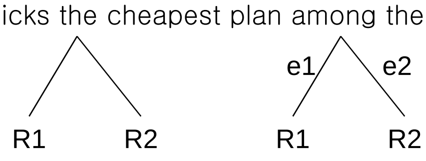


Optimization Algorithm With Complete Rank-Ordering

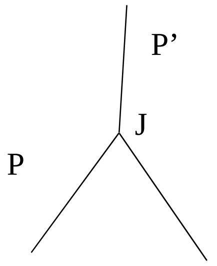
- Guarantees optimality
- Polynomial in number of user-defined predicates but exponential in number of relations
- No exhaustive enumeration of join space
- Complete rank-ordering rule reduces number of tags
- Pruning rules help compare plans with different tags

Approximate Algorithms

- Known algorithms do badly
  - Traditional algorithm (all predicates pushed-down)
  - Pullup (all predicate evaluations deferred)
  - PullRank [Hellerstein 94]
    - Considers all possible placements of expensive predicates locally either immediately preceding or immediately after join
    - Picks the cheapest plan among them



Conservative Local Heuristic



Pick two local plans:

a) Minimize costs of evaluating P and J

b) Minimize costs of evaluating P, P'& J

Distinguish between Pull-up and Push-down but blur the distinctions among tags

Conservative Local Heuristic

- The two local plans favor locally pushing down or pulling over expensive selections
- It is now possible for the optimizer to consider more alternatives.

Conservative Local Heuristic

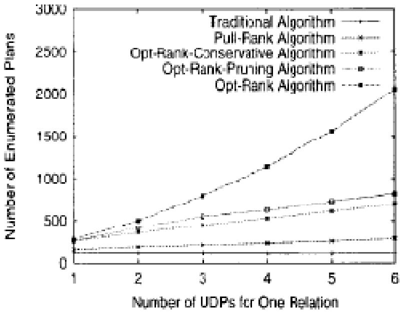
- Guarantees Optimality in many important cases
  - Single Join
  - Single Predicate
  - Pullover/Pushdown is the optimal
- Near optimal performance and significantly better than other known approximate methods

Experimental Studies

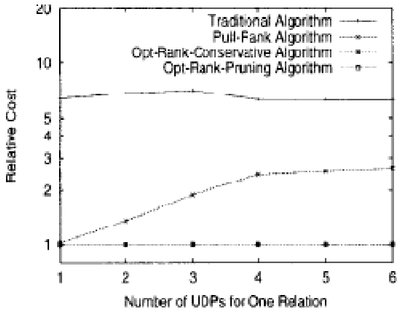
- Implemented by extending a System R style prototype
- Varied two parameters
  - Number of user-defined predicates
  - Distribution of predicates among relations
- Setup:
  - Varied number of distinct values and relation cardinality
  - Popular indexing structures and join methods

Experimental Results

100 Random Queries



4 Join Query



4 Joins Query

67

Experimental Results

- Plans by traditional optimizers and PullRank heuristic fared badly
- Pruning strategies are very effective
- Conservative Local Heuristic generates plans close to optimal with almost no overhead
- Distribution of predicates have interesting effect on the optimization cost

Sep 3, 1996 68

Query Optimization with Foreign Functions

Query:

```
select business.name, map.location
from business, map
where business.type = 'Restaurant'
      and business.etakid = map.etakid
      and inside(w, map.location)
      and business.earning > expected_revenue(bisiness.size)
```

Rewrite Rule:

```
Insode(w1, point), Inside(w2, point)
          -> Inside(w, point), Intersect(w1,w2,w)
```

69

Foreign Functions in Query Optimization [VLDB 1993]

- A Query Q, a set of rewrite rules R and a set of base tables B:
- Question 1: What are all the alternative ways of answering Q?
- Question 2: How can we pick the best execution plan for Q from its alternatives?

70

Materialized Views

- emp(name, salary, dno)
- dept(dno, mgr, floor, location)

Query:

```
select name
from emp, dept
where emp.sal > 220k
    and dept.floor=1 and emp.dno = dept.dno
```

View:

```
create view emp_loc(name, size, location) as
select name, size, location
from emp, dept
where emp.dno = dept.dno
```

71

Materialized Views in Query Optimization [ICDE 1995]

- A Query Q, a set of materialized views V and a set of base tables B:
- Question 1: What are all the alternative ways of answering Q?
- Question 2: How can we pick the best execution plan for Q from its alternatives?

72

Including Group-By in Query Optimization [VLDB 1994]

- emp(name, salary, dno)
- dept(dno, mgr, floor, location)

```
select emp.dno, sum(emp.salary)
from emp, dept
where emp.dno = dept.dno and
      dept.floor = 5
group by dno
```

Group-By(dno)

Join(dno)

emp dept

73

Including Group-By in Query Optimization

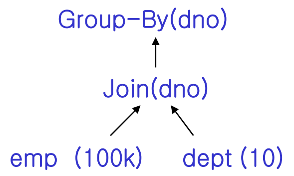
- Traditional Execution
  - A Two-phase execution
    - Execute all joins
    - Then, process group-by
  - Observation:
    - Execution plans that interleave join and group-by may be much cheaper

74

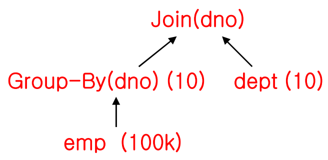
Including Group-By in Query Optimization

- emp(name, salary, dno)
- dept(dno, mgr, floor, location)

- A Traditional Execution



- An Alternative Execution



75

Advantage of Early Group-by

- May reduce the cost of a join by reducing the size of input relation significantly
- May allow the use of indexes over base tables to combine scan and group-by

76

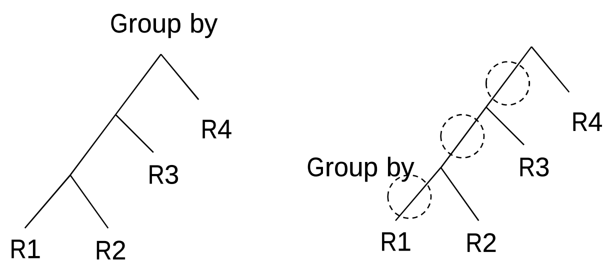
Including Group-By in Query Optimization

- Transformations (push group-by past join)
  - Invariant grouping
  - Simple Coalescing grouping
  - May not always desirable
- Query Optimization
  - Integration with System R style optimizer needs to be considered
  - But search space is too large
  - Propose a greedy conservative heuristic

77

Invariant Grouping

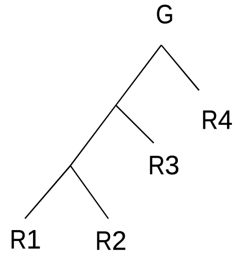
- Substitutes group-by with an early group-by
- Take advantage of foreign key
- Universally applicable for any aggregate function

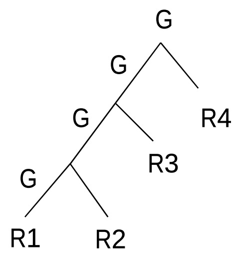


78

Simple Coalescing Grouping

- early group-bys are added
- Future join needs not be with foreign keys
- Exploit the property of aggregate functions





79

Multiple Query Optimization [DKE 1994]

- emp(name, salary, dno)
- dept(dno, mgr, floor, location)

Query1:

```
select emp.name
from emp, dept
where emp.dno = dept.dno
            and dept.floor = 1
```

Query2:

```
select emp.name
from emp, dept
where emp.dno = dept.dno
            and dept.floor = 1
          and emp.age < 30
```

80

Parametric Query Optimization [VLDB1992]

- emp(name, salary, dno)
- dept(dno, mgr, floor, location)

Query:

```
select emp.name
from emp, dept
where emp.dno = dept.dno
    and dept.floor = 1
    and emp.age < X
```