Query optimization

Lecture 7
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

13 May 2015

Admin

- Project part 2 due today
  - Part 3 has been posted

Admin

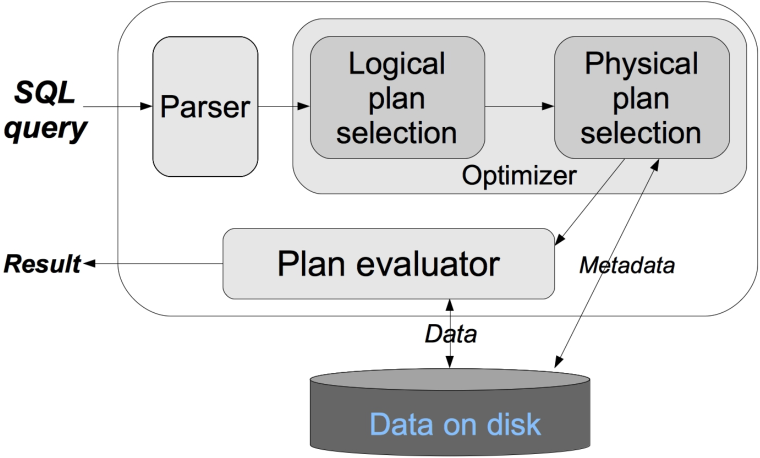
- Project part 2 due today
  - Part 3 has been posted
- No lecture this Friday or next Wednesday
- Written individual assignment will be posted this week

Last time

Size estimation (heuristics, histograms)

View creation, maintenance, and use

The life of a query

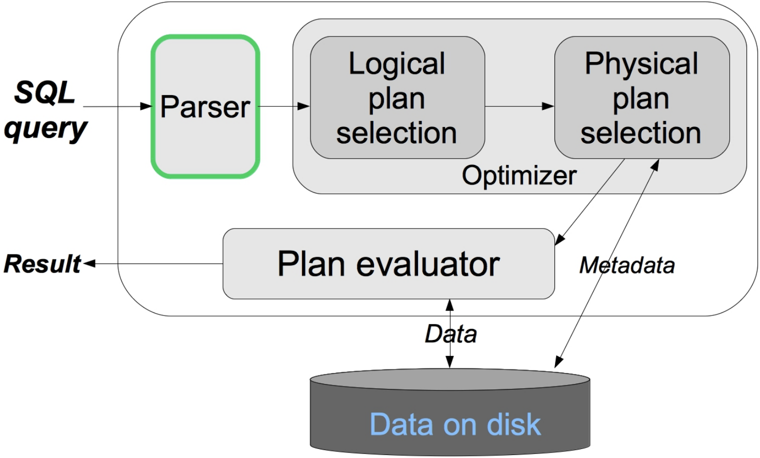


Today

Query compilation

- parsing
- logical optimization
- physical optimization

The life of a query: parsing



The life of a query: parsing

- Employee(EID, Name, ECity)
- Company(CID, Name, CCity)
- WorksFor(EID, CID, Salary)

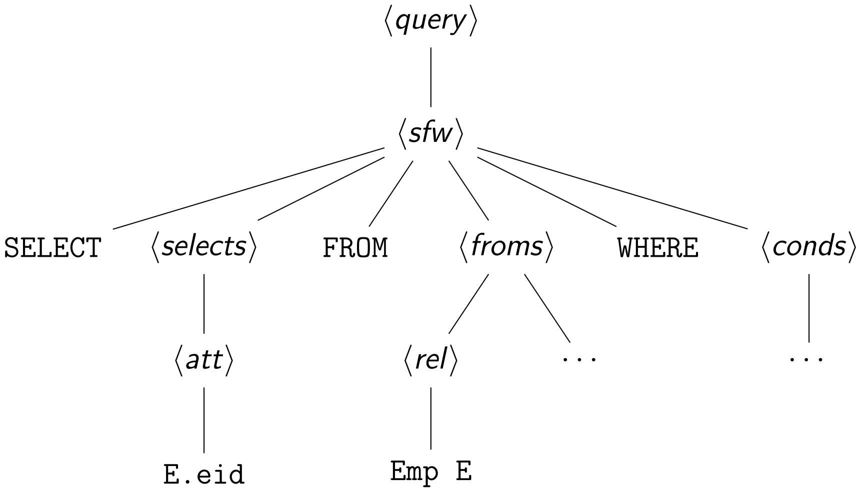
What are the ID's of employees living and working in Best with above-average salaries?

The life of a query: parsing

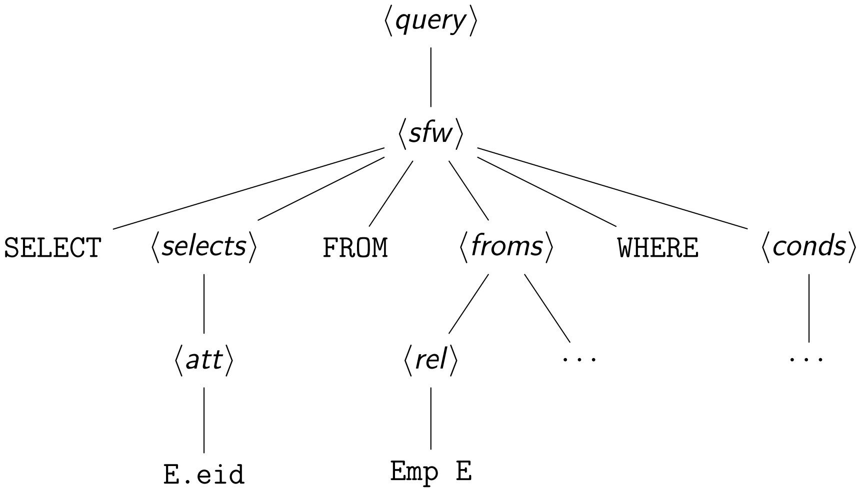
- Employee(EID, Name, ECity)
- Company(CID, Name, CCity)
- WorksFor(EID, CID, Salary)

```
SELECT E.EID
FROM Employee E, WorksFor W, Company C
WHERE E.EID = W.EID AND W.CID = C.CID
      AND E.ECity = ‘Best’
      AND C.CCity = ‘Best’
      AND W.Salary > (SELECT AVG(SALARY)
                      FROM WorksFor)
```

The life of a query: parsing



The life of a query: parsing



... and preprocess for semantics (e.g., typing)

The life of a query: parsing

Parser generates a collection of query *blocks*

- block = a S-F-W query with no nesting
  - essentially, a conjunctive query

The life of a query: parsing

Parser generates a collection of query *blocks*

- block = a S-F-W query with no nesting
  - essentially, a conjunctive query
- typically focus on optimizing one block at a time

The life of a query: parsing

- Employee(EID, Name, ECity)
- Company(CID, Name, CCity)
- WorksFor(EID, CID, Salary)

```
SELECT E.EID
FROM Employee E, WorksFor W, Company C
WHERE E.EID = W.EID AND W.CID = C.CID
      AND E.ECity = ‘Best’
      AND C.CCity = ‘Best’
      AND W.Salary >
                (SELECT AVG(SALARY)
                 FROM WorksFor)
```

nested block

The life of a query: parsing

- Employee(EID, Name, ECity)
- Company(CID, Name, CCity)
- WorksFor(EID, CID, Salary)

```
SELECT E.EID
FROM Employee E, WorksFor W, Company C
WHERE E.EID = W.EID AND W.CID = C.CID
      AND E.ECity = 'Best'
      AND C.CCity = 'Best'
      AND W.Salary >
                (SELECT AVG(SALARY)
                 FROM WorksFor)
```

outer block

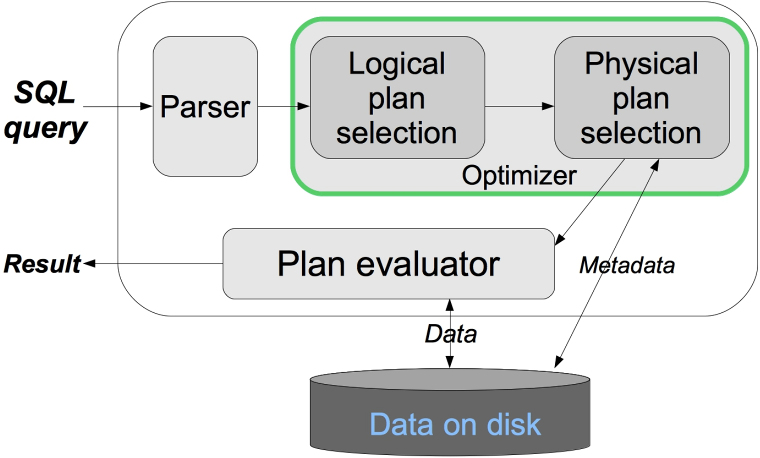
The life of a query: parsing

- Employee(EID, Name, ECity)
- Company(CID, Name, CCity)
- WorksFor(EID, CID, Salary)

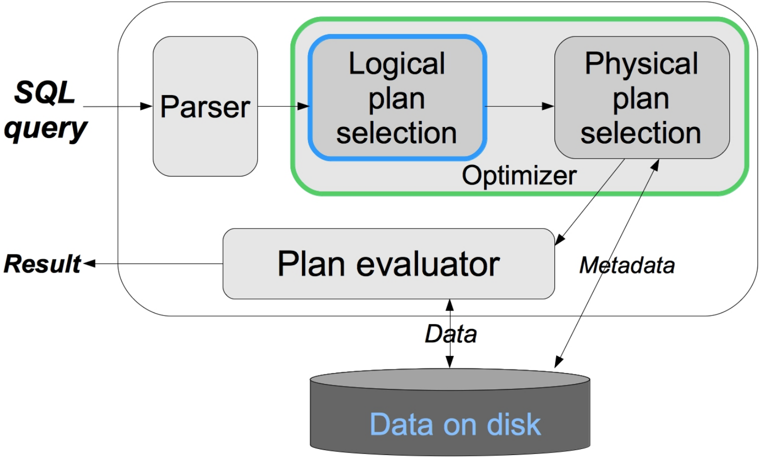
```
SELECT E.EID
FROM Employee E, WorksFor W, Company C
WHERE E.EID = W.EID AND W.CID = C.CID
      AND E.ECity = ‘Best’
      AND C.CCity = ‘Best’
      AND W.Salary >
                (SELECT AVG(SALARY)
                 FROM WorksFor)
```

focus on optimizing each block separately

The life of a query: optimization



The life of a query: logical optimization



The life of a query: logical optimization

Goal: map a query-block to a “preferred” logical query plan

The life of a query: logical optimization

Goal: map a query-block to a "preferred" logical query plan

Step A: map the block's parse tree to equivalent RA expression, working bottom-up

1. map FROM list to cartesian product of all relations

The life of a query: logical optimization

Goal: map a query-block to a "preferred" logical query plan

Step A: map the block's parse tree to equivalent RA expression, working bottom-up

1. map FROM list to cartesian product of all relations
2. form single σ for WHERE conditions

The life of a query: logical optimization

Goal: map a query-block to a "preferred" logical query plan

Step A: map the block's parse tree to equivalent RA expression, working bottom-up

1. map FROM list to cartesian product of all relations
2. form single σ for WHERE conditions
3. form π list from the SELECT clause

The life of a query: logical optimization

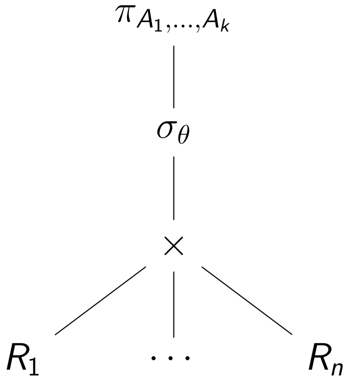
Goal: map a query-block to a “preferred” logical query plan

Step A: map the block’s parse tree to equivalent RA expression, working bottom-up

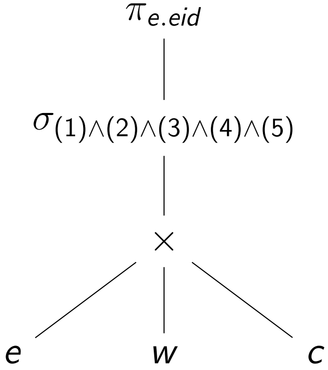
1. map FROM list to cartesian product of all relations
2. form single σ for WHERE conditions
3. form π list from the SELECT clause

any aggregation can be applied after this $\pi \cdot \sigma \cdot \times$

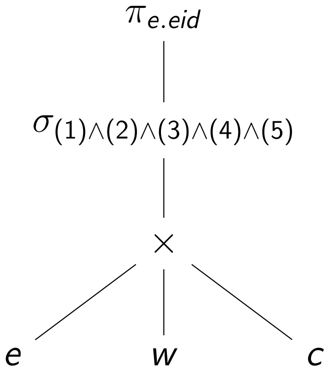
The life of a query: logical optimization



The life of a query: logical optimization



The life of a query: logical optimization



(1) $e.eid = w.eid$
(2) $w.cid = c.cid$
(3) $e.ecity = \text{`Best'}$
(4) $c.ccity = \text{`Best'}$
(5) $w.salary > \mathcal{V}$

The life of a query: logical optimization

Step B: improve this initial logical query plan (i.e., generate a preferred logical query plan)

The life of a query: logical optimization

Step B: improve this initial logical query plan (i.e., generate a preferred logical query plan)

based on equivalence rules, used to identify different ways of formulating the query

The life of a query: logical optimization

Step B: improve this initial logical query plan (i.e., generate a preferred logical query plan)

based on equivalence rules, used to identify different ways of formulating the query

this defines a search space of alternative plans, to be considered by the optimizer

RA equivalence rules: commutativity & associativity

$$\begin{aligned} R \star S &= S \star R \\ (R \star S) \star T &= R \star (S \star T) \end{aligned}$$

for $\star \in \{\times, \bowtie, \cup, \cap\}$

RA equivalence rules: commutativity & associativity

So, we have for example

$$\begin{aligned} R \cap S &= S \cap R \\ (R \cap S) \cap T &= R \cap (S \cap T) \end{aligned}$$

RA equivalence rules: commutativity & associativity

So, we have for example

$$\begin{aligned} R \cap S &= S \cap R \\ (R \cap S) \cap T &= R \cap (S \cap T) \end{aligned}$$

example application.

$$\begin{aligned} &\sigma_\theta(\pi_A(R)) \cap (\pi_A(S) \cap \pi_A(T)) \\ &\qquad = (\sigma_\theta(\pi_A(R)) \cap \pi_A(S)) \cap \pi_A(T) \end{aligned}$$

RA equivalence rules: selection laws

splitting

$$\sigma_{C_1 \wedge \cdots \wedge C_n}(R) \;=\; \sigma_{C_1}(\cdots \sigma_{C_n}(R) \cdots)$$

RA equivalence rules: selection laws

splitting

$$
\begin{aligned}
\sigma_{C_1 \wedge \cdots \wedge C_n}(R) &= \sigma_{C_1}(\cdots \sigma_{C_n}(R) \cdots) \\
\sigma_{C_1 \vee C_2}(R) &= \sigma_{C_1}(R) \cup \sigma_{C_2}(R)
\end{aligned}
$$

RA equivalence rules: selection laws

splitting

$$\begin{aligned} \sigma_{C_1 \wedge \cdots \wedge C_n}(R) &= \sigma_{C_1}(\cdots \sigma_{C_n}(R) \cdots) \\ \sigma_{C_1 \vee C_2}(R) &= \sigma_{C_1}(R) \cup \sigma_{C_2}(R) \end{aligned}$$

example application.

$\sigma_{a=1}(\sigma_{b<c}(R)) \cup \sigma_{a=3}(\sigma_{b<c}(R))$

RA equivalence rules: selection laws

splitting

$$\begin{aligned}
\sigma_{C_1 \wedge \cdots \wedge C_n}(R) &= \sigma_{C_1}(\cdots \sigma_{C_n}(R) \cdots) \\
\sigma_{C_1 \vee C_2}(R) &= \sigma_{C_1}(R) \cup \sigma_{C_2}(R)
\end{aligned}$$

example application.

$$\sigma_{a=1}(\sigma_{b<c}(R)) \cup \sigma_{a=3}(\sigma_{b<c}(R)) \;=\; \sigma_{a=1 \vee a=3}(\sigma_{b<c}(R))$$

RA equivalence rules: selection laws

splitting

$$\begin{aligned} \sigma_{C_1 \wedge \cdots \wedge C_n}(R) &= \sigma_{C_1}(\cdots \sigma_{C_n}(R) \cdots) \\ \sigma_{C_1 \vee C_2}(R) &= \sigma_{C_1}(R) \cup \sigma_{C_2}(R) \end{aligned}$$

example application.

$$\begin{aligned} \sigma_{a=1}(\sigma_{b<c}(R)) \cup \sigma_{a=3}(\sigma_{b<c}(R)) &= \sigma_{a=1 \vee a=3}(\sigma_{b<c}(R)) \\ &= \sigma_{(a=1 \vee a=3) \wedge b<c}(R) \end{aligned}$$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) \;=\; \sigma_\theta(R) \cup \sigma_\theta(S)$$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) \;=\; \sigma_\theta(R) \cup \sigma_\theta(S)$$

difference

$$\sigma_\theta(R - S) \;=\; \sigma_\theta(R) - \sigma_\theta(S)$$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) \;=\; \sigma_\theta(R) \cup \sigma_\theta(S)$$

difference

$$\begin{aligned}\sigma_\theta(R - S) &= \sigma_\theta(R) - \sigma_\theta(S)\\ &= \sigma_\theta(R) - S\end{aligned}$$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) \;=\; \sigma_\theta(R) \cup \sigma_\theta(S)$$

difference

$$\begin{aligned}\sigma_\theta(R - S) &= \sigma_\theta(R) - \sigma_\theta(S)\\ &= \sigma_\theta(R) - S\end{aligned}$$

and similarly for $\times, \bowtie, \cap$

RA equivalence rules: selection laws

union

$$\sigma_\theta(R \cup S) \;=\; \sigma_\theta(R) \cup \sigma_\theta(S)$$

difference

$$\begin{aligned} \sigma_\theta(R - S) &= \sigma_\theta(R) - \sigma_\theta(S) \\ &= \sigma_\theta(R) - S \end{aligned}$$

and similarly for $\times, \bowtie, \cap$

proof of the difference rule

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$\sigma_{(a=1 \vee a=3) \wedge b<c}(R \bowtie S)$

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\sigma_{(a=1 \vee a=3) \wedge b<c}(R \bowtie S) \;=\; \sigma_{a=1 \vee a=3}(\sigma_{b<c}(R \bowtie S))$$

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\begin{aligned}
\sigma_{(a=1 \vee a=3) \wedge b<c}(R \bowtie S) &= \sigma_{a=1 \vee a=3}(\sigma_{b<c}(R \bowtie S)) \\
&= \sigma_{a=1 \vee a=3}(R \bowtie \sigma_{b<c}(S))
\end{aligned}$$

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\begin{aligned}
\sigma_{(a=1 \vee a=3) \wedge b<c}(R \bowtie S) &= \sigma_{a=1 \vee a=3}(\sigma_{b<c}(R \bowtie S)) \\
&= \sigma_{a=1 \vee a=3}(R \bowtie \sigma_{b<c}(S)) \\
&= \sigma_{a=1 \vee a=3}(R) \bowtie \sigma_{b<c}(S).
\end{aligned}$$

RA equivalence rules: selection laws

example application. for $R(a, b)$ and $S(b, c)$

$$\begin{aligned}
\sigma_{(a=1 \vee a=3) \wedge b<c}(R \bowtie S) &= \sigma_{a=1 \vee a=3}(\sigma_{b<c}(R \bowtie S)) \\
&= \sigma_{a=1 \vee a=3}(R \bowtie \sigma_{b<c}(S)) \\
&= \sigma_{a=1 \vee a=3}(R) \bowtie \sigma_{b<c}(S).
\end{aligned}$$

note how this brings the selection conditions closer to the input relations ...

RA equivalence rules: projection law

if, for each $1 \leqslant i \leqslant n$, we have $a_i \subseteq a_{i+1}$, for subsets $a_1, \ldots, a_n$ of $atts(R)$, then

$$\pi_{a_1}(R) \;=\; \pi_{a_1}(\pi_{a_2}(\cdots(\pi_{a_n}(R))\cdots)).$$

RA equivalence rules: projection law

if, for each $1 \leqslant i \leqslant n$, we have $a_i \subseteq a_{i+1}$, for subsets $a_1, \ldots, a_n$ of $atts(R)$, then

$$\pi_{a_1}(R) \;=\; \pi_{a_1}(\pi_{a_2}(\cdots(\pi_{a_n}(R))\cdots)).$$

example application. for $R(a, b, c)$

$$\pi_a(R) \;=\; \pi_a(\pi_{a,b}(R)).$$

RA equivalence rules

Prove or disprove: $\pi_a(R - S) = \pi_a(R) - \pi_a(S)$,
where a is a nonempty set of attributes in R and S.

RA equivalence rules

Prove or disprove: $\pi_a(R - S) = \pi_a(R) - \pi_a(S)$, where a is a nonempty set of attributes in R and S.

counterexample. Consider $R(a, b)$ and $S(a, b)$, with respective instances $r = \{(1, 2)\}$ and $s = \{(1, 3)\}$. Then $\pi_a(r - s) = \{(1)\}$, but $\pi_a(r) - \pi_a(s) = \{\}$.

RA equivalence rules

Prove or disprove: $\pi_x(R \bowtie S) = \pi_x(\pi_{xy}(R) \bowtie S)$,
where $y = atts(R) \cap atts(S)$ and $x \subseteq atts(R)$.

RA equivalence rules

Prove or disprove: $\pi_x(R \bowtie S) = \pi_x(\pi_{xy}(R) \bowtie S)$, where $y = atts(R) \cap atts(S)$ and $x \subseteq atts(R)$.

proof. We show that $\pi_x(R \bowtie S) \subseteq \pi_x(\pi_{xy}(R) \bowtie S)$. The other direction is similar.

Suppose that $t \in \pi_x(R \bowtie S)$. Then we have that (1) there exists $t' \in R \bowtie S$ such that $t = t'[x]$ (i.e., t' projected on x).

It then follows that there exists $r \in R$ and $s \in S$ such that (2) $t'[atts(R)] = r$, (3) $t'[atts(S)] = s$, and (4) $r[y] = s[y]$.

From (2) we have that (5) $t'[xy] \in \pi_{xy}(R)$.

From (2), (4), and (5) we have that (6) $t'[xy] \bowtie s \in \pi_{xy}(R) \bowtie S$.

From (1) and (6), we have that $t = t'[x] \in \pi_x(\pi_{xy}(R) \bowtie S)$, as desired.

RA equivalence rules: SPJ laws

1. if $atts(\theta) \subseteq \{a_1, \ldots, a_n\}$, then

$$\pi_{\{a_1,\ldots,a_n\}}(\sigma_\theta(R)) \;=\; \sigma_\theta(\pi_{\{a_1,\ldots,a_n\}}(R))$$

RA equivalence rules: SPJ laws

1. if $atts(\theta) \subseteq \{a_1, \ldots, a_n\}$, then

$$\pi_{\{a_1,\ldots,a_n\}}(\sigma_\theta(R)) \;=\; \sigma_\theta(\pi_{\{a_1,\ldots,a_n\}}(R))$$

2. $R \bowtie_\theta S = \sigma_\theta(R \times S)$

RA equivalence rules: SPJ laws

3. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, and $a_2 \subseteq atts(S)$, then

$$\pi_a(R \times S) \;=\; \pi_{a_1}(R) \times \pi_{a_2}(S)$$

RA equivalence rules: SPJ laws

3. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, and $a_2 \subseteq atts(S)$, then

$$\pi_a(R \times S) \;=\; \pi_{a_1}(R) \times \pi_{a_2}(S)$$

4. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, $a_2 \subseteq atts(S)$, and $atts(\theta) \subseteq a$, then

$$\pi_a(R \bowtie_\theta S) \;=\; \pi_{a_1}(R) \bowtie_\theta \pi_{a_2}(S)$$

RA equivalence rules: SPJ laws

3. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, and $a_2 \subseteq atts(S)$, then

$$\pi_a(R \times S) \;=\; \pi_{a_1}(R) \times \pi_{a_2}(S)$$

4. if $a_1 \cup a_2 = a$, $a_1 \subseteq atts(R)$, $a_2 \subseteq atts(S)$, and $atts(\theta) \subseteq a$, then

$$\pi_a(R \bowtie_\theta S) \;=\; \pi_{a_1}(R) \bowtie_\theta \pi_{a_2}(S)$$

exercise. prove 4

RA equivalence rules

Some heuristics

1. push down selections (σ) as far as they can go, splitting conjunctions in conditions ($\wedge$)

RA equivalence rules

Some heuristics

1. push down selections (σ) as far as they can go, splitting conjunctions in conditions ($\wedge$)
    - single most important strategy

RA equivalence rules

Some heuristics

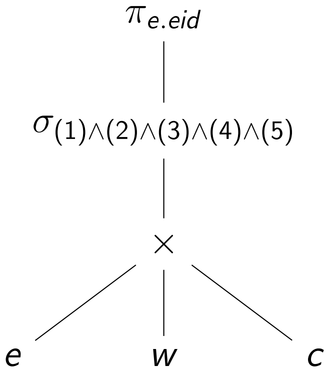
1. push down selections (σ) as far as they can go, splitting conjunctions in conditions ($\wedge$)
   - single most important strategy
2. push down projections (π)

RA equivalence rules

Some heuristics

1. push down selections (σ) as far as they can go, splitting conjunctions in conditions ($\wedge$)
   - single most important strategy
2. push down projections (π)
3. if possible, turn $\sigma + \times$ into $\bowtie$, which is generally much cheaper to evaluate than σ and $\times$ evaluated separately

RA equivalence rules

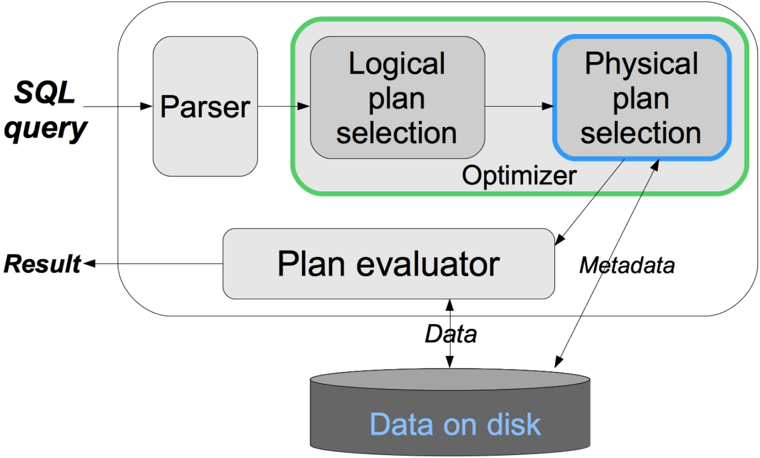


$e(eid, name, ecity)$
$c(cid, name, ccity)$
$w(eid, cid, salary)$

(1) $e.eid = w.eid$
(2) $w.cid = c.cid$
(3) $e.ecity = \text{`Best'}$
(4) $c.ccity = \text{`Best'}$
(5) $w.salary > \mathcal{V}$

let's logically optimize our query, under these heuristics

The life of a query: physical optimization



The life of a query: physical optimization

Decisions

- two crucial decisions the optimizer makes
  - the order in which the physical operators are applied on the input relations
  - the choice of physical algorithms for each node of the plan
- of course, these are not independent

The life of a query: physical optimization

cost-based query optimization

- enumerate alternative plans, estimate the cost of each plan, pick the plan with minimum cost
- access methods and join algorithms define a search space
  - this space can be huge!
  - plan enumeration is the exploration of this search space

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider “linear” joins

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider "linear" joins

- there are then $5! = 120$ possible plans

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider "linear" joins

- there are then $5! = 120$ possible plans
- if we add one additional access method, we then have $2^5 \cdot 5! = 3840$ possible plans

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider "linear" joins

- there are then $5! = 120$ possible plans
- if we add one additional access method, we then have $2^5 \cdot 5! = 3840$ possible plans
- if we add one additional join algorithm, we then have $2^4 \cdot 2^5 \cdot 5! = 61440$ possible plans

The life of a query: physical optimization

For example, suppose we have five relations, only one access method, only one join algorithm, and we only consider "linear" joins

- there are then $5! = 120$ possible plans
- if we add one additional access method, we then have $2^5 \cdot 5! = 3840$ possible plans
- if we add one additional join algorithm, we then have $2^4 \cdot 2^5 \cdot 5! = 61440$ possible plans

In general, with n relations, there are $\frac{(2(n-1))!}{(n-1)!}$ different join orders. For $n = 5$, this gives us 860,160 possible plans with two access methods and join algorithms. For $n = 10$, this gives us roughly 17.6 billion plans ...

The life of a query: physical optimization

cost-based query optimization

- exhaustive search is certainly out of the question
- it could be that exploring the search space might take longer than actually evaluating the query

The life of a query: physical optimization

cost-based query optimization

- exhaustive search is certainly out of the question
- it could be that exploring the search space might take longer than actually evaluating the query
- the manner in which the plan space is explored describes a (physical) query optimization method
  - dynamic programming, rule-based optimization, randomized search, ...

The life of a query: physical optimization

cost-based query optimization

- costing plans and exploring a plan space is nontrivial
- now, consider that the DBMS must do this for 1000's of queries, simultaneously!
- hence, all of this must be done quickly, without looking back

The life of a query: physical optimization

cost-based query optimization

- costing plans and exploring a plan space is nontrivial
- now, consider that the DBMS must do this for 1000's of queries, simultaneously!
- hence, all of this must be done quickly, without looking back
- query optimization is very much still an active area of research
- indeed, rarely will an optimizer find the optimal plan
  - it must, however, not pick a bad plan – just an OK one

Query optimization: dynamic-programming algorithm

```
procedure FindBestPlan(S)
    if (bestplan[S].cost ≠ ∞) /* bestplan[S] already computed */
        return bestplan[S]
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost based on best way of accessing S
    else for each non-empty subset S1 of S such that S1 ≠ S
        P1 = FindBestPlan(S1)
        P2 = FindBestPlan(S − S1)
        A = best algorithm for joining results of P1 and P2
        cost = P1.cost + P2.cost + cost of A
        if cost < bestplan[S].cost
            bestplan[S].cost = cost
            bestplan[S].plan = "execute P1.plan; execute P2.plan;
                                join results of P1 and P2 using A"
    return bestplan[S]
```

Query optimization: dynamic-programming algorithm

```
procedure FindBestPlan(S)
    if (bestplan[S].cost ≠ ∞) /* bestplan[S] already computed */
        return bestplan[S]
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost based on best way of accessing S
    else for each non-empty subset S1 of S such that S1 ≠ S
        P1 = FindBestPlan(S1)
        P2 = FindBestPlan(S − S1)
        A = best algorithm for joining results of P1 and P2
        cost = P1.cost + P2.cost + cost of A
        if cost < bestplan[S].cost
            bestplan[S].cost = cost
            bestplan[S].plan = "execute P1.plan; execute P2.plan;
                                join results of P1 and P2 using A"
    return bestplan[S]
```

Running time is $\mathcal{O}(3^n)$, which gives us roughly 59,000 plans for $n = 10$ (compare this with 17.6 billion plans)

Query optimization: dynamic-programming algorithm

```
procedure FindBestPlan(S)
    if (bestplan[S].cost ≠ ∞) /* bestplan[S] already computed */
        return bestplan[S]
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost based on best way of accessing S
    else for each non-empty subset S1 of S such that S1 ≠ S
        P1 = FindBestPlan(S1)
        P2 = FindBestPlan(S − S1)
        A = best algorithm for joining results of P1 and P2
        cost = P1.cost + P2.cost + cost of A
        if cost < bestplan[S].cost
            bestplan[S].cost = cost
            bestplan[S].plan = "execute P1.plan; execute P2.plan;
                                join results of P1 and P2 using A"
    return bestplan[S]
```

- most widely used strategy
- works well for queries with fewer than 10 to 15 joins

Query optimization in System R

- follows the classic dynamic programming approach

Query optimization in System R

- follows the classic dynamic programming approach
- heuristics: use the equivalence rules to push down selections and projections, and delay cartesian products

Query optimization in System R

- follows the classic dynamic programming approach
- heuristics: use the equivalence rules to push down selections and projections, and delay cartesian products
- constraints: left-deep plans, nested-loops and sort-merge join only

Query optimization in System R

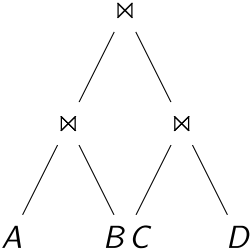
- follows the classic dynamic programming approach
- heuristics: use the equivalence rules to push down selections and projections, and delay cartesian products
- constraints: left-deep plans, nested-loops and sort-merge join only
  - left-deep plans facilitate pipelining of output of each operator into the next operator, without materializing intermediate result

Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$

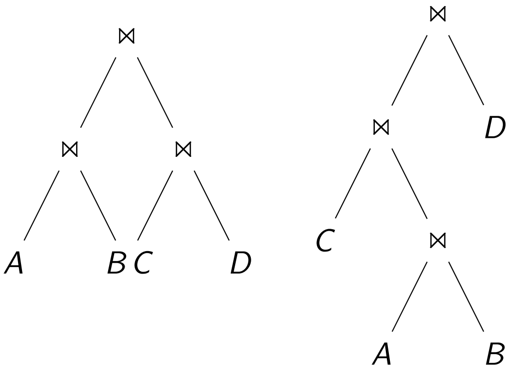
Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$



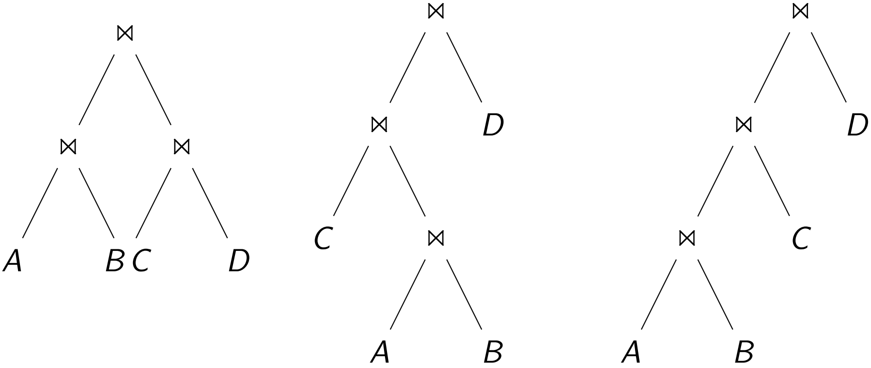
Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$



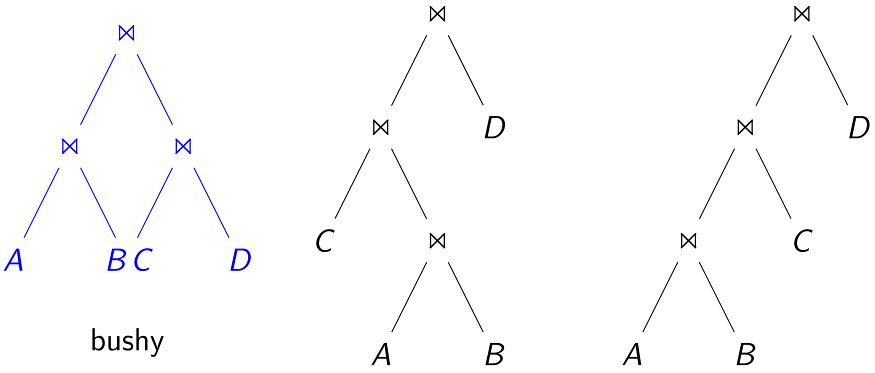
Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$



Query optimization in System R

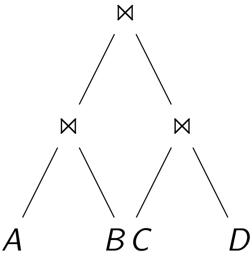
$$A \bowtie B \bowtie C \bowtie D$$

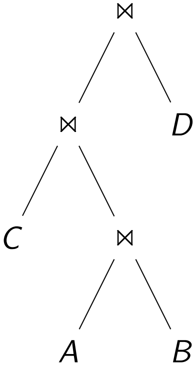


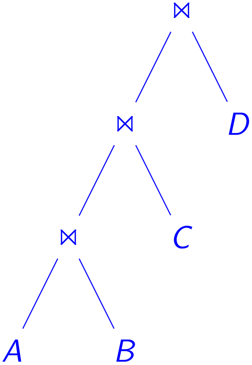
bushy

Query optimization in System R

$$A \bowtie B \bowtie C \bowtie D$$







left-deep

Query optimization in System R

- left-deep plans differ only in the order of relations, the access method (file-scan or index) for each relation, and the join method for each join

Query optimization in System R

- left-deep plans differ only in the order of relations, the access method (file-scan or index) for each relation, and the join method for each join
- in spite of pruning, the resulting search space is still exponential in the number of relations (as we saw earlier)
  - actually is $\mathcal{O}(n2^n)$ in practice
  - with $n = 10$, this is roughly 10,000 plans (compared with 59,000 and 17,600,000,000)

Query optimization in System R

enumeration of left-deep plans: basic idea

- identify cheapest way to access each single relation in the query

Query optimization in System R

enumeration of left-deep plans: basic idea

- identify cheapest way to access each single relation in the query
- for every access method and join predicate, find the cheapest way to join in a second relation

Query optimization in System R

enumeration of left-deep plans: basic idea

- identify cheapest way to access each single relation in the query
- for every access method and join predicate, find the cheapest way to join in a second relation
- ...

Query optimization in System R

enumeration of left-deep plans, for N relations

- Pass 1: find cheapest 1-relation plan for each relation
- Pass 2: find cheapest way to join result of each 1-relation plan (as outer) to another relation (output: all 2-relation plans)
- ...
- Pass N: find cheapest way to join result of a $(N-1)$-relation plan (as outer) to the Nth relation (output: all N-relation plans)

for each subset of relations, retain only cheapest plan overall, using $\mathcal{O}(2^N)$ space

Query optimization in System R

Suppose in our running example that we have 102 buffer pages available, and

- The Emp table has 10,000 tuples, on 1000 pages, with a clustered B-tree index on EID.
- The Works table has 20,000 tuples, on 2000 pages, sorted on $\langle$EID, CID$\rangle$.
- The Company table has 10,000 tuples, on 1000 pages, with a clustered B-tree index on CID.

Query optimization in System R

Pass 1: finding cheapest 1-relation plans

- for Emp, we have the local predicate $e.ecity = \text{`Best'}$. However, with only an index on EID, the cheapest plan is a file-scan (with on-the fly application of the selection predicate

Query optimization in System R

Pass 1: finding cheapest 1-relation plans

- for Emp, we have the local predicate $e.ecity = \text{`Best'}$. However, with only an index on EID, the cheapest plan is a file-scan (with on-the fly application of the selection predicate
- likewise, we have file-scans as cheapest access method for Works, and Company

Query optimization in System R

Pass 2: finding cheapest 2-relation plans

- we don't consider $\{E, C\}$ since this is just a cartesian product

Query optimization in System R

Pass 2: finding cheapest 2-relation plans

- we don't consider $\{E, C\}$ since this is just a cartesian product
- for EW (or WE), we can perform merge-join at cost $1000 + 2000 = 3000$
- for WE, we can perform index nested-loop-join at cost $2000 + 3 \cdot 20,000 = 62,000$
- for EW, we can perform block-nested-loops-join at cost $1000 + 2000 \cdot (1000/100) = 21,000$

Query optimization in System R

Pass 2: finding cheapest 2-relation plans

- for WC, we can perform index nested-loop-join at cost $2000 + 3 \cdot 20,000 = 62,000$
- for WC (or CW), we can perform sort-merge-join at cost $4000 + 2000 + 1000 = 7000$
- for CW, we can perform block-nested-loops-join at cost $1000 + 2000 \cdot 10 = 21,000$

Query optimization in System R

Pass 2: finding cheapest 2-relation plans

- for WC, we can perform index nested-loop-join at cost $2000 + 3 \cdot 20,000 = 62,000$
- for WC (or CW), we can perform sort-merge-join at cost $4000 + 2000 + 1000 = 7000$
- for CW, we can perform block-nested-loops-join at cost $1000 + 2000 \cdot 10 = 21,000$

So, best cost for $\{E, W\}$ is 3000, and for $\{W, C\}$ is 7000

Query optimization in System R

Pass 3: finding cheapest 3-relation plans

- for EW joined with C, assuming 2000 pages in result of EW, we can perform sort merge-join at cost $2000 + 4000 + 1000 = 7000$

Query optimization in System R

Pass 3: finding cheapest 3-relation plans

- for EW joined with C, assuming 2000 pages in result of EW, we can perform sort merge-join at cost $2000 + 4000 + 1000 = 7000$
- for WC joined with E, assuming 2000 pages in result of WC, we can perform sort merge-join at cost $2000 + 4000 + 1000 = 7000$

The life of a query: logical optimization

So, best overall plan is

- a merge-join of Emp and Works (with local predicates applied)
- followed by a sort-merge-join with Company,
- followed by the final projection

at cost $3000 + 7000 = 10000$ I/Os

Summary

- Query optimizer is at the heart of the query engine
- the paradigm typically followed is cost-based optimization
- System R follows a dynamic programming approach to plan space search
  - other practical approaches include randomized (e.g., simulated annealing) and genetic algorithms

Summary

- Query optimizer is at the heart of the query engine
- the paradigm typically followed is cost-based optimization
- System R follows a dynamic programming approach to plan space search
  - other practical approaches include randomized (e.g., simulated annealing) and genetic algorithms

Next time (Friday 22 May): distributed data management