CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #19: Query Optimization
Notes

- Material NOT in the book!

- Some parts from (a copy of the paper is on the course webpage)
  
Select * From Blah B Where B.blah = blah

Usually there is a heuristics-based rewriting step before the cost-based steps.
Multiple Algorithms: Range Searches

- Sequential Scan
- Hashes
- B-Trees
- ....

- Saw some of them in previous lecture
Multiple Algorithms: Joins

- Merge-Join (like merge-sort)
- Hash-Join (using hashes)
- Indexed-Join (using indexes)
- Nested loops Join (most obvious)
- ...

- We haven’t covered them in class
Why Query optimization?

- SQL: ~declarative
- good q-opt -> big difference
  - eg., seq. Scan vs
  - B-tree index, on P=1,000 pages

- We had some ‘manual q-opt’ in Project Assignment 3 ➔ too much effort!
Q-opt steps

- bring query in internal form (eg., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best
Q-opt - example

```
select name
from STUDENT, TAKES
where c-id = '4604' and
STUDENT.ssn = TAKES.ssn
```
Q-opt - example

Canonical form

STUDENT  TAKES  STUDENT  TAKES

\[\pi \sigma\]

Prakash 2013  VT CS 4604
Q-opt - example

Hash join; merge join; nested loops; 

\( \pi \) 

\( \sigma \) \( \rightarrow \) Index; seq scan 

STUDENT \hspace{1cm} TAKES
Equivalence of expressions

- A.k.a.: syntactic q-opt
- in short: perform selections and projections early
Equivalence of expressions

Q: How to prove a transformation rule?

\[ \sigma_P(R1 \bowtie R2) = \sigma_P(R1) \bowtie \sigma_P(R2) \]

A: use RA, to show that LHS = RHS, eg:

\[ \sigma_P(R1 \cup R2) = \sigma_P(R1) \cup \sigma_P(R2) \]
Equivalence of expressions

\[ \sigma_p(R1 \cup R2) = \sigma_p(R1) \cup \sigma_p(R2) \]

\[ t \in LHS \iff \]

\[ t \in (R1 \cup R2) \land P(t) \iff \]

\[ (t \in R1 \lor t \in R2) \land P(t) \iff \]

\[ (t \in R1 \land P(t)) \lor (t \in R2 \land P(t)) \iff \]
Equivalence of expressions

\[ \sigma_P(R1 \cup R2) = \sigma_P(R1) \cup \sigma_P(R2) \]

...  

\[(t \in R1 \land P(t)) \lor (t \in R2 \land P(t)) \iff (t \in \sigma_P(R1)) \lor (t \in \sigma_P(R2)) \iff t \in \sigma_P(R1) \cup \sigma_P(R2) \iff t \in RHS \]

QED
Equivalence of expressions

Q: how to disprove a rule??

\[ \pi_A(R1 - R2) = \pi_A(R1) - \pi_A(R2) \]

Construct a counter-example!
Equivalence of expressions

- Selections
  - perform them early
  - break a complex predicate, and push
    $$\sigma_{p_1 \land p_2 \land \ldots \land p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots \sigma_{p_n}(R))\ldots)$$
  - simplify a complex predicate
    - (‘X=Y and Y=3’) -> ‘X=3 and Y=3’
Equivalence of expressions

- Projections
  - perform them early (but carefully...)
    - Smaller tuples
    - Fewer tuples (if duplicates are eliminated)
  - project out all attributes except the ones requested or required (e.g., joining attr.)
Equivalence of expressions

- Joins
  - Commutative, associative
  \[
  R \Join S = S \Join R \\
  (R \Join S) \Join T = R \Join (S \Join T)
  \]
  - Q: n-way join - how many diff. orderings?
Equivalence of expressions

- Joins - Q: n-way join - how many diff. orderings?
- A: Catalan number \( \sim 4^n \)
  - Exhaustive enumeration: too slow.

Prakash 2013
(Some) Transformation Rules (1)

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.
\[ \sigma_{\theta_1 \land \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E)) \]

2. Selection operations are commutative.
\[ \sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E)) \]

3. Only the last in a sequence of projection operations is needed, the others can be omitted.
\[ \Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{L_n}(E))\ldots)) = \Pi_{L_1}(E) \]

4. Selections can be combined with Cartesian products and theta joins.
   a. \[ \sigma_{\theta}(E_1 \times E_2) = E_1 \Join_{\theta} E_2 \]
   b. \[ \sigma_{\theta_1}(E_1 \Join_{\theta_2} E_2) = E_1 \Join_{\theta_1 \land \theta_2} E_2 \]
5. Theta-join operations (and natural joins) are commutative.

\[ E_1 \Join_{\theta} E_2 = E_2 \Join_{\theta} E_1 \]

6. (a) Natural join operations are associative:

\[ (E_1 \Join E_2) \Join E_3 = E_1 \Join (E_2 \Join E_3) \]

(b) Theta joins are associative in the following manner:

\[ (E_1 \Join_{\theta_1} E_2) \Join_{\theta_2 \land \theta_3} E_3 = E_1 \Join_{\theta_1 \land \theta_3} (E_2 \Join_{\theta_2} E_3) \]

where \( \theta_2 \) involves attributes from only \( E_2 \) and \( E_3 \).
7. The selection operation distributes over the theta join operation under the following two conditions:

(a) When all the attributes in $\theta_0$ involve only the attributes of one of the expressions $(E_1)$ being joined.

$$\sigma_{\theta_0}(E_1 \Join_\theta E_2) = (\sigma_{\theta_0}(E_1)) \Join_\theta E_2$$

(b) When $\theta_1$ involves only the attributes of $E_1$ and $\theta_2$ involves only the attributes of $E_2$.

$$\sigma_{\theta_1 \land \theta_2}(E_1 \Join_\theta E_2) = (\sigma_{\theta_1}(E_1)) \Join_\theta (\sigma_{\theta_2}(E_2))$$
Q-opt steps

- bring query in internal form (eg., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best
Cost-based Query Sub-System

Queries

Select *
From Blah B
Where B.blah = blah

Usually there is a heuristics-based rewriting step before the cost-based steps.

Query Parser

Query Optimizer

Plan Generator

Plan Cost Estimator

Catalog Manager

Schema

Statistics

Query Plan Evaluator

Prakash 2013
Cost estimation

- Eg., find ssn’s of students with an ‘A’ in 4604 (using seq. scanning)
- How long will a query take?
  - CPU (but: small cost; decreasing; tough to estimate)
  - Disk (mainly, # block transfers)
- How many tuples will qualify?
- (what statistics do we need to keep?)
Cost estimation

- Statistics: for each relation ‘r’ we keep
  - \( nr : \# \) tuples;
  - \( Sr : \) size of tuple in bytes
Cost estimation

- Statistics: for each relation ‘r’ we keep
  - ...  
  - $V(A,r)$: number of distinct values of attr. ‘A’  
  - (recently, histograms, too)
Derivable statistics

- blocking factor = max# records/block (=??
- br: # blocks (=??
- SC(A,r) = selection cardinality = avg# of records with A=given (=??

\[ \text{Sr} \]

\[ \begin{array}{c}
\text{fr} \\
\hline
#1 \\
#2 \\
\vdots \\
\hline
#br
\end{array} \]
Derivable statistics

- blocking factor = max# records/block (= B/Sr; B: block size in bytes)
- br: # blocks (= nr / (blocking-factor) )
Derivable statistics

- $SC(A,r) = \text{selection cardinality} = \text{avg# of records with } A=\text{given} \ (= \frac{nr}{V(A,r)}) \ (\text{assumes uniformity...})$

eg: 10,000 students, 10 departments – how many students in CS?
Additional quantities we need:

- For index ‘i’:
  - $f_i$: average fanout (~50-100)
  - $H_{Ti}$: # levels of index ‘i’ (~2-3)
    - $\sim \log(#\text{entries})/\log(f_i)$
  - $L_{Bi}$: # blocks at leaf level
Statistics

- Where do we store them?
- How often do we update them?
Q-opt steps

- bring query in internal form (e.g., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best
Selections

- we saw simple predicates (A=constant; eg., ‘name=Smith’)
- how about more complex predicates, like
  - ‘salary > 10K’
  - ‘age = 30 and job-code=“analyst”’
- what is their selectivity?
Selections – complex predicates

- selectivity \( \text{sel}(P) \) of predicate \( P \) :
  - \( \text{sel}(P) = \frac{SC(P)}{nr} \)
  - \( \text{sel}(P) \) is the fraction of tuples that qualify
Selections – complex predicates

- eg., assume that \( V(\text{grade}, \text{TAKES})=5 \) distinct values
- simple predicate \( P: A=\text{constant} \)
  - \( \text{sel}(A=\text{constant}) = 1/V(A,r) \)
  - eg., \( \text{sel} (\text{grade}='B') = 1/5 \)
- (what if \( V(A,r) \) is unknown??)
Selections – complex predicates

- range query: \( \text{sel}(\text{grade} \geq 'C') \)
  
  \( \text{sel}(A>a) = \frac{(A_{\text{max}} - a)}{(A_{\text{max}} - A_{\text{min}})} \)
Selections - complex predicates

- negation: sel( grade != ‘C’) 
  - sel( not P) = 1 – sel(P) 
  - (Observation: selectivity =~ probability)
Selections - complex predicates

- Conjunction:
  - `sel( grade = ‘C’ and course = ‘4604’ )`
  - `sel(P1 and P2) = sel(P1) * sel(P2)`
  - INDEPENDENCE ASSUMPTION
Selections - complex predicates

- Disjunction:
  - \( \text{sel}( \text{grade} = 'C' \text{ or } \text{course} = '4604') \)
  - \( \text{sel}(P1 \text{ or } P2) = \text{sel}(P1) + \text{sel}(P2) - \text{sel}(P1 \text{ and } P2) \)
  - \( \text{sel}(P1) + \text{sel}(P2) - \text{sel}(P1) \ast \text{sel}(P2) \)
  - INDEPENDENCE ASSUMPTION, again

Prakash 2013
Selections - complex predicates

- disjunction: in general
  - \(\text{sel}(\text{P1 or P2 or } \ldots \text{ Pn}) = 1 - (1 - \text{sel}(\text{P1})) \times (1 - \text{sel}(\text{P2})) \times \ldots \times (1 - \text{sel}(\text{Pn}))\)
Selections – summary

- sel(A=constant) = 1/V(A,r)
- sel( A>a) = (Amax – a) / (Amax – Amin)
- sel(not P) = 1 – sel(P)
- sel(P1 and P2) = sel(P1) * sel(P2)
- sel(P1 or P2) = sel(P1) + sel(P2) – sel(P1)*sel(P2)
- sel(P1 or ... or Pn) = 1 - (1-sel(P1))*...*(1-sel(Pn))

- UNIFORMITY and INDEPENDENCE ASSUMPTIONS
Result Size Estimation for Joins

- **Q:** Given a join of R and S, what is the range of possible result sizes (in #of tuples)?
  - Hint: what if $R_{cols} \cap S_{cols} = \emptyset$?
  - $R_{cols} \cap S_{cols}$ is a key for R (and a Foreign Key in S)?
Result Size Estimation for Joins

- General case: $R_{cols} \cap S_{cols} = \{A\}$ (and $A$ is key for neither)
  - match each $R$-tuple with $S$-tuples
    \[
    \text{est\_size} \sim N\text{Tuples}(R) \times N\text{Tuples}(S) / N\text{Keys}(A, S) \\
    \sim nr \times ns / V(A, S)
    \]
  - symmetrically, for $S$:
    \[
    \text{est\_size} \sim N\text{Tuples}(R) \times N\text{Tuples}(S) / N\text{Keys}(A, R) \\
    \sim nr \times ns / V(A, R)
    \]
  - Overall:
    \[
    \text{est\_size} = N\text{Tuples}(R) \times N\text{Tuples}(S) / \text{MAX}\{N\text{Keys}(A, S), N\text{Keys}(A, R)\}
    \]
On the Uniform Distribution Assumption

- Assuming uniform distribution is rather crude
Histograms

- For better estimation, use a histogram

Equiwidth histogram

Equidepth histogram ~ quantiles

Prakash 2013
Q-opt Steps

- bring query in internal form (e.g., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
  - single relation
  - multiple relations
- estimate cost; pick best
plan generation

- Selections – eg.,
  
  ```sql
  select *
  from TAKES
  where grade = 'A'
  ```

- Plans?

...
plan generation

- Plans?
  - seq. scan
  - binary search
    - (if sorted & consecutive)
  - index search
    - if an index exists
plan generation

seq. scan – cost?
- $br$ (worst case)
- $br/2$ (average, if we search for primary key)
plan generation

binary search – cost?

if sorted and consecutive:
  - $\sim \log(\text{br}) +$
  - $\text{SC}(A,r)/\text{fr}$ (=blocks spanned by qual. tuples)
plan generation

estimation of selection cardinalities $SC(A,r)$:

**non-trivial** — we saw it earlier
plan generation

method#3: index – cost?
– Tricky
Q-opt Steps

- bring query in internal form (eg., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
  - single relation
  - multiple relations
- estimate cost; pick best
n-way joins

- $r_1 \text{ JOIN } r_2 \text{ JOIN } ... \text{ JOIN } r_n$
- typically, break problem into 2-way joins
  - choose between NL, sort merge, hash join, ...
Queries Over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly → need to restrict search space

- Fundamental decision in System R (IBM): only left-deep join trees are considered. Advantages?

Prakash 2013
Queries Over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly → *need to restrict search space*

- Fundamental decision in System R (IBM): *only left-deep join trees* are considered. Advantages?
  - *fully pipelined* plans.
    - Intermediate results not written to temporary files.

---

Prakash 2013

VT CS 4604
Queries over Multiple Relations

- Enumerate the orderings (= left deep tree)
- Enumerate the plans for each operator
- Enumerate the access paths for each table

Dynamic programming, to save cost estimations
Candidate Plans

1. Enumerate relation orderings:

```
1. S B R
2. S R B
3. B S R
4. B R S
5. S B R
6. S R B
7. B S R
8. B R S
```

Prune plans with cross-products immediately!

---

SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B

2. Enumerate **join algorithm** choices:

+ do same for 4 other plans

→ 4*4 = 16 plans so far..
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B

3. Enumerate access method choices:

+ do same for other plans
Now estimate the **cost** of each plan

Example:

```
  B (heap scan)
 /    \
NLJ   R (INDEX scan on R.sid)
 /     \
NLJ   S (heap scan)
```

Prakash 2013
Query Re-writing

- Re-write nested queries
- to: de-correlate and/or flatten them
Correlated vs Uncorrelated

- The previous subqueries did not depend on anything outside the subquery
  - ...and thus need to be executed just once.
  - These are called **uncorrelated**.

- A **correlated** subquery depends on data from the outer query
  - ... and thus has to be executed for each row of the outer table(s)
Example: Decorrelating a Query

SELECT CourseName, Enrollment
FROM Courses
WHERE EXISTS
(SELECT *
FROM Teaches T
WHERE (T.name = 'Smith')
AND (Courses.num = T.num));

Equivalent uncorrelated query:
SELECT CourseName, Enrollment
FROM Courses
WHERE Courses.Num IN
(SELECT T.num
FROM Teaches T
WHERE T.name = 'Smith')

- **Advantage**: nested block only needs to be executed **once** (rather than once per S tuple)
Example: “Flattening” a Query

```
SELECT CourseName, Enrol
FROM Courses
WHERE Courses.Num IN
    (SELECT T.num
     FROM Teaches T
     WHERE T.name = 'Smith')
```

**Equivalent non-nested query:**

```
SELECT CourseName, Enrol
FROM Courses C, Teaches T
WHERE Courses.Num = T.num
AND T.name = 'Smith'
```

- **Advantage:** can use a join algorithm + optimizer can select among join algorithms & reorder freely
Conclusions

- Ideas to remember:
  - syntactic q-opt – do selections early
  - selectivity estimations (uniformity, indep.; histograms; join selectivity)
  - left-deep joins
    - dynamic programming
  - handling correlated sub-queries