Overview - detailed

- Why q-opt?
- Equivalence of expressions
- Cost estimation
- Plan generation
- Plan evaluation

Cost-based Query Sub-System

- Usually there is a heuristics-based rewriting step before the cost-based steps.
Why Q-opt?

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

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

```sql
select name
from STUDENT, TAKES
where c-id='415' and
STUDENT.ssn=TAKES.ssn
```
Overview - detailed

• Why q-opt?
• Equivalence of expressions
• Cost estimation
  • ...

Q-opt - example

STUDENT 
\( \pi \)
\( \sigma \)
TAKES

Canonical form

STUDENT 
\( \pi \)
\( \sigma \)
TAKES

Q-opt - example

STUDENT 
\( \pi \)
\( \sigma \)
TAKES

Hash join;
merge join;
nested loops;

Index; seq scan

Nested loops;
Equivalence of expressions

• A.k.a.: syntactic q-opt
• in short: perform selections and projections early
• More details: see transf. rules in text

Q: How to prove a transf. rule?

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

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

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

$\sigma_x(R1 \cup R2) = \sigma_x(R1) \cup \sigma_x(R2)$

... $t \in R1 \land P(t)) \lor (t \in R2 \land P(t)) \iff$

$t \in \sigma_x(R1)) \lor (t \in \sigma_x(R2)) \iff$

$t \in \sigma_x(R1) \cup \sigma_x(R2) \iff$

$t \in \text{RHS}$

QED

Equivalence of expressions

- Q: how to disprove a rule??

$\pi_y(R1 - R2) \gamma \pi_y(R1) - \pi_y(R2)$

Equivalence of expressions

- Q: how to disprove a rule??

$\pi_y(R1 - R2) \gamma \pi_y(R1) - \pi_y(R2)$

<table>
<thead>
<tr>
<th>R1</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Smith</td>
<td>pizza</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Smith</td>
<td>steak</td>
</tr>
</tbody>
</table>
Equivalence of expressions

• Selections
  – perform them early
  – break a complex predicate, and push
  $$\sigma_{p_1 \land p_2 \land \ldots \land p_k}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots \sigma_{p_k}(R))\ldots)$$
  – simplify a complex predicate
    • \((X=Y \land Y=3) \Rightarrow X=3 \land Y=3\)

• 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.)

• Joins
  – Commutative, associative
  \(R \bowtie S = S \bowtie R\)
  \((R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)\)
  – 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.

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

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

- Eg., find ssn’s of students with an ‘A’ in 415 (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
  - V(A,r): number of distinct values of attr. ‘A’
  - (histograms, too)
Derivable statistics

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

Derivable statistics

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

Derivable statistics

- SC(A,r) = selection cardinality = avg# of records with A=given (= nr / V(A,r))
(assumes uniformity...) – eg: 10,000 students, 10 colleges – how many students in SCS?
Additional quantities we need:

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

Statistics

- Where do we store them?
- How often do we update them?

Q-opt steps

- bring query in internal form (eg., 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 sel(P) of predicate P :
  == fraction of tuples that qualify
  sel(P) = SC(P) / nr

Selections – complex predicates

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

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

- negation: \( \text{sel}(\text{grade} \neq 'C') \)
  \[ \text{sel}(\text{not } P) = 1 - \text{sel}(P) \]
  (Observation: selectivity \( \approx \) probability)

- conjunction: \( \text{sel}(\text{grade} = 'C' \text{ and } \text{course} = '415') \)
  \[ \text{sel}(P_1 \text{ and } P_2) = \text{sel}(P_1) \times \text{sel}(P_2) \]
  (INDEPENDENCE ASSUMPTION)
Selections – complex predicates

disjunction:
- \( \text{sel( grade = 'C' or course = '415') } \)
- \( \text{sel}(P_1 \text{ or } P_2) = \text{sel}(P_1) + \text{sel}(P_2) - \text{sel}(P_1 \text{ and } P_2) \)
- \( = \text{sel}(P_1) + \text{sel}(P_2) - \text{sel}(P_1)\ast\text{sel}(P_2) \)
- INDEPENDENCE ASSUMPTION, again

Selections – complex predicates

disjunction: in general
\[
\text{sel}(P_1 \text{ or } P_2 \text{ or } ... \text{ or } P_n) = \frac{1}{V(A,r)} \]
\[
1 - (1 - \text{sel}(P_1)) \ast (1 - \text{sel}(P_2)) \ast ... \ast (1 - \text{sel}(P_n))
\]

Selections – summary
- \( \text{sel}(A=\text{constant}) = \frac{1}{V(A,r)} \)
- \( \text{sel}(A>a) = \frac{(\text{Amax} - a)}{\text{Amax} - \text{Amin}} \)
- \( \text{sel}(\text{not } P) = 1 - \text{sel}(P) \)
- \( \text{sel}(P_1 \text{ and } P_2) = \text{sel}(P_1) \ast \text{sel}(P_2) \)
- \( \text{sel}(P_1 \text{ or } P_2) = \text{sel}(P_1) + \text{sel}(P_2) - \text{sel}(P_1)\ast\text{sel}(P_2) \)
- \( \text{sel}(P_1 \text{ or } ... \text{ or } P_n) = 1 - (1 - \text{sel}(P_1))\ast...\ast(1 - \text{sel}(P_n)) \)
- 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)?

\[
\text{nr} \leq \text{ns}
\]
Result Size Estimation for Joins

- General case: \( R_{\text{cols}} \cap S_{\text{cols}} = \{A\} \) (and A is key for neither)

  Hint: for a given tuple of R, how many tuples of S will it match?

  - match each R-tuple with S-tuples
    \[ \text{est_size} \sim \frac{\text{NTuples}(R) \cdot \text{NTuples}(S)}{\text{NKeys}(A, S)} \]
  - symmetrically, for S:
    \[ \text{est_size} \sim \frac{\text{NTuples}(R) \cdot \text{NTuples}(S)}{\text{NKeys}(A, R)} \]
  - Overall:
    \[ \text{est_size} = \frac{\text{NTuples}(R)^2}{\max\{\text{NKeys}(A, S), \text{NKeys}(A, R)\}} \]

On the Uniform Distribution Assumption

- Assuming uniform distribution is rather crude

Distribution D

Uniform distribution approximating D
Histograms

• For better estimation, use a histogram

Equidepth histogram

Equiwidth histogram

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 – e.g.,
  select *
  from TAKES
  where grade = ‘A’
• Plans?
plan generation

- Plans?
  - seq. scan
  - binary search
    - if sorted & consecutive
  - index search
    - if an index exists

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

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

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

- non-trivial – we saw it earlier

$\begin{array}{c}
\text{fr} \\
\hline
\text{Sr} \\
\hline
#1 \\
#2 \\
\ldots \\
#br
\end{array}$

plan generation

method #3: index – cost?
- levels of index +
- blocks w/ qual. tuples

- case #1: primary key
- case #2: sec. key – clustering index
- case #3: sec. key – non-clust. index

plan generation

method #3: index – cost?
- levels of index +
- blocks w/ qual. tuples

- case #1: primary key – cost: $HT_i + 1$
plan generation

method#3: index – cost?
- levels of index +
- blocks w/ qual. tuples

case#2: sec. key – clustering index
HTi + SC(A,r)/fr

... #fr #Sr

... #1 #2

... #br

Faloutsos
CMU SCS 15-415

plan generation

method#3: index – cost?
- levels of index +
- blocks w/ qual. tuples

case#3: sec. key – non-clustering index
HTi + SC(A,r)
(actually, pessimistic...)

Faloutsos
CMU SCS 15-415

plan generation

method#3: index – cost?
- levels of index +
- blocks w/ qual. tuples

(actually, pessimistic...
better estimates: Cardenas' formula)

Faloutsos
CMU SCS 15-415
Cardena’s formula

- \( q \): # qual records
- \( Q \): # qual. blocks
- \( N \): # records total
- \( B \): # blocks total

- \( Q = ? \)

Pessimistic:
- \( Q = q \)

More realistic:
- \( Q = q \) if \( q \leq B \)
- \( Q = B \) otherwise
Cardena’s formula

- Cardenas’ formula

\[ Q = B \left[ 1 - \left(1 - \frac{1}{B} \right)^q \right] \]
Cardena’s formula

- Cardenas’ formula

\[ Q = B \left[ 1 - (1 - 1/B)^q \right] \]

Prob(it avoids it, q times)

#1

#2

... #B

Cardena’s formula

- Cardenas’ formula

\[ Q = B \left[ 1 - (1 - 1/B)^q \right] \]

Prob(our favorite block is hit at least once, after q selections)

#1

#2

... #B

Plans for single relation - summary

- no index: scan (dup-elim; sort)
- with index:
  - single index access path
  - multiple index access path
  - sorted index access path
  - index-only access path
Overview - detailed

- Why q-opt?
- Equivalence of expressions
- Cost estimation
- Plan generation
- Plan evaluation

Citation


Frequently cited database publications

http://www.informatik.uni-trier.de/~ley/db/about/top.html

<table>
<thead>
<tr>
<th>#</th>
<th>Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>371</td>
<td>Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, Thomas G. Price: Access Path Selection in a Relational Database Management System. SIGMOD Conference 1979: 23-34</td>
</tr>
</tbody>
</table>
Statistics for Optimization

- NCARD(T) - cardinality of relation T in tuples
- TCARD(T) - number of pages containing tuples from T
- \(P(T) = \frac{TCARD(T)}{\# \text{ of non-empty pages in the segment}}\)
  - If segments only held tuples from one relation there would be no need for \(P(T)\)
- ICARD(I) - number of distinct keys in index I
- NINDX(I) - number of pages in index I

Predicate Selectivity Estimation

<table>
<thead>
<tr>
<th>Condition</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr = value</td>
<td>(F = \frac{1}{ICARD(\text{attr index})}) if index exists (F = \frac{1}{10}) otherwise</td>
</tr>
<tr>
<td>attr1 = attr2</td>
<td>(F = \frac{1}{\max(ICARD(I1),ICARD(I2))}) or (F = \frac{1}{ICARD(I)}) if only index i exists, or (F = \frac{1}{10})</td>
</tr>
<tr>
<td>val1 &lt; attr &lt; val2</td>
<td>(F = \frac{(\text{value2-value1})}{(\text{high key-low key})}) (F = \frac{1}{4}) otherwise</td>
</tr>
<tr>
<td>expr1 or expr2</td>
<td>(F = F(\text{expr1})+F(\text{expr2})-F(\text{expr1})\times F(\text{expr2}))</td>
</tr>
<tr>
<td>expr1 and expr2</td>
<td>(F = F(\text{expr1}) \times F(\text{expr2}))</td>
</tr>
<tr>
<td>NOT expr</td>
<td>(F = 1 - F(\text{expr}))</td>
</tr>
</tbody>
</table>

Costs per Access Path Case

<table>
<thead>
<tr>
<th>Condition</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique index matching equal predicate</td>
<td>1+1+W</td>
</tr>
<tr>
<td>Clustered index I matching &gt;=1 preds</td>
<td>(F(\text{preds})\times(\text{NINDX(I)}+\text{TCARD})+W\times \text{RSICARD})</td>
</tr>
<tr>
<td>Non-clustered index I matching &gt;=1 preds</td>
<td>(F(\text{preds})\times(\text{NINDX(I)}+\text{NCARD})+W\times \text{RSICARD})</td>
</tr>
<tr>
<td>Segment scan</td>
<td>(\text{TCARD}/P + W\times \text{RSICARD})</td>
</tr>
</tbody>
</table>
Q-opt steps

- bring query in internal form (eg., parse tree)
- … into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
  - single relation
  - multiple relations
    - Main idea
    - Dynamic programming – reminder
    - Example
- estimate cost; pick best

n-way joins

- r1 JOIN r2 JOIN ... JOIN rn
- 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 \( \rightarrow \) need to restrict search space
- Fundamental decision in System R: only left-deep join trees are considered. Advantages?
Queries Over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly \(\rightarrow\) need to restrict search space
- Fundamental decision in System R: only left-deep join trees are considered. Advantages?
  - fully pipelined plans.
    - Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).

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

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
    - Main idea
    - Dynamic programming – reminder
    - Example
- estimate cost; pick best
Cheapest flight PIT -> SG?

Assumption: NO package deals: cost CDG->SG is always $800, no matter how reached CDG

Solution: compute partial optimal, left-to-right:
(Reminder: Dynamic Programming)

Solution: compute partial optimal, left-to-right:

[Diagram with city connections and costs]

Solution: compute partial optimal, left-to-right:

[Diagram with city connections and costs]

Solution: compute partial optimal, left-to-right:

[Diagram with city connections and costs]
So, best price is $1,500 – which legs?

A: follow the winning edges, backwards
So, best price is $1,500 – which legs?
A: follow the winning edges, backwards

Q: what are the states, costs and arrows, in q-opt?
A: set of intermediate result tables
Q-opt and Dyn. Programming

- E.g., compute \( R \) join \( S \) join \( T \)

- Details: how to record the fact that, say \( R \) is sorted on \( R.a \)? or that the user requires sorted output?

- A:
  - E.g., consider the query
  ```sql
  select *
  from \( R \), \( S \), \( T \)
  where \( R.a = S.a \) and \( S.b = T.b \)
  order by \( R.a \)
  ```

- A: record orderings, in the state
  - E.g., consider the query
  ```sql
  select *
  from \( R \), \( S \), \( T \)
  where \( R.a = S.a \) and \( S.b = T.b \)
  order by \( R.a \)
  ```
Q-opt and Dyn. Programming

- E.g., compute \( R \Join S \Join T \) order by \( R.a \)

\[
\begin{align*}
150 \text{ (SM)} & : R \Join S \\
2,500 \text{ (NL)} & : S \Join T \\
\end{align*}
\]

Any other changes?

- E.g., compute \( R \Join S \Join T \) order by \( R.a \)

\[
\begin{align*}
150 \text{ (SM)} & : R \Join S \\
2,500 \text{ (NL)} & : S \Join T \\
\end{align*}
\]

Any other changes?

- E.g., compute \( R \Join S \Join T \) order by \( R.a \)

\[
\begin{align*}
150 \text{ (SM)} & : R \Join S \\
2,500 \text{ (NL)} & : S \Join T \\
\end{align*}
\]
Q-opt steps

• bring query in internal form (eg., parse tree)
• ... into ‘canonical form’ (syntactic q-opt)
• generate alt. plans
  – single relation
  – multiple relations
  • Main idea
  • Dynamic programming – reminder
  • Example
• estimate cost; pick best

Candidate Plans
1. Enumerate relation orderings:

![Diagram showing relation orderings]

Prune plans with cross-products immediately!

Candidate Plans
2. Enumerate join algorithm choices:

![Diagram showing join algorithm choices]

+ do same for 4 other plans
\[ \times 4 = 16 \text{ plans so far} \]
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:

Q-opt steps

• bring query in internal form (eg., parse tree)
• … into ‘canonical form’ (syntactic q-opt)
• generate alt. plans
  – single relation
  – multiple relations
  – nested subqueries
• estimate cost; pick best
Q-opt steps

• Everything so far: about a single query block

Query Rewriting

• Re-write nested queries
• to: de-correlate and/or flatten them

Example: Decorrelating a Query

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

Equivalent uncorrelated query:
SELECT S.sid  
FROM Sailors S  
WHERE S.sid IN  
(SELECT R.sid  
FROM Reserves R  
WHERE R.bid=103)  

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

```sql
SELECT S.sid
FROM Sailors S
WHERE S.sid IN
(SELECT R.sid
FROM Reserves R
WHERE R.bid=103)
```

Equivalent non-nested query:

```sql
SELECT S.sid
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
AND R.bid=103
```

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

Structure of query optimizers:

System R:
- break query in query blocks
- simple queries (i.e., no joins): look at stats
- n-way joins: left-deep join trees; i.e., only one intermediate result at a time
  - pros: smaller search space; pipelining
  - cons: may miss optimal
- 2-way joins: NL and sort-merge

More heuristics by Oracle, Sybase and Starburst (→ DB2)

In general: q-opt is very important for large databases.

(`explain select <sql-statement>’ gives plan)
Q-opt steps

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

Conclusions

- Ideas to remember:
  - syntactic q-opt – do selections early
  - selectivity estimations (uniformity, indep.; histograms; join selectivity)
  - hash join (nested loops; sort-merge)
  - left-deep joins
  - dynamic programming