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
```
Q-opt - example

Canonical form

STUDENT TAKES

Q-opt - example

Canonical form

STUDENT TAKES

Q-opt - example

Hash join; merge join; nested loops;

Index; seq scan

STUDENT TAKES
Overview - detailed

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

Equivalence of expressions

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

Equivalence of expressions

- Q: How to prove a transf. rule?
  \[ \sigma_p(R1 \bowtie \alpha R2) = \sigma_p(R1) \bowtie \alpha \sigma_p(R2) \]
- A: use RTC, to show that LHS = RHS, eg:
  \[ \sigma_p(R1 \cup R2) = \sigma_p(R1) \cup \sigma_p(R2) \]
Equivalence of expressions

\[ \sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2) \]
\[ t \in LHS \iff \]
\[ (t \in R_1 \cup R_2) \land P(t) \iff \]
\[ (t \in R_1 \lor t \in R_2) \land P(t) \iff \]
\[ (t \in R_1 \land P(t)) \lor (t \in R_2 \land P(t)) \iff \]

Equivalence of expressions

\[ \sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2) \]
\[ ... \]
\[ (t \in R_1 \land P(t)) \lor (t \in R_2 \land P(t)) \iff \]
\[ (t \in \sigma_p(R_1)) \lor (t \in \sigma_p(R_2)) \iff \]
\[ t \in \sigma_p(R_1) \cup \sigma_p(R_2) \iff \]
\[ t \in RHS \]
\[ QED \]

Equivalence of expressions

- Q: how to disprove a rule??

\[ \pi_x(R_1 - R_2) = \pi_x(R_1) - \pi_x(R_2) \]
Equivalence of expressions

• Q: how to disprove a rule??

\[ \pi_A(R_1 - R_2) = \pi_A(R_1) - \pi_A(R_2) \]

R1 | A | B  
---|---|---
    | Smith | pizza

R2 | A | B  
---|---|---
    | Smith | steak

Selections

– perform them early
– break a complex predicate, and push
  \[ \sigma_{\rho_1 \cdot \rho_2 \cdots \rho_k}(R) = \sigma_{\rho_1}(\sigma_{\rho_2}(\cdots \sigma_{\rho_k}(R))\cdots) \]
– simplify a complex predicate
  \[ (X=Y \text{ and } Y=3) \rightarrow (X=3 \text{ and } Y=3) \]

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.)
Equivalence of expressions

- 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?

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 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

Cost estimation

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

Cost estimation
Cost estimation

- Statistics: for each relation 'r' we keep
  - ... 
  - $V(A,r)$: 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

- 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’:
  - fi: average fanout (~50-100)
  - HTi: # levels of index ‘i’ (~2-3)
    - \( \log (#\text{entries}) / \log(f) \)
  - LBi: # 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; e.g., '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 :
  \[ \text{fraction of tuples that qualify} \]
  \[ \text{sel}(P) = \frac{SC(P)}{nr} \]
Selections – complex predicates

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

\[
\begin{array}{c|c|c}
\text{grade} & \text{count} \\
\hline
F & A \\
\end{array}
\]

Selections – complex predicates

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

\[
\begin{array}{c|c|c}
\text{grade} & \text{count} \\
\hline
F & A \\
\end{array}
\]

Selections - complex predicates

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

conjunction:
- \( \text{sel( grade = 'C' and course = '415') } \)
- \( \text{sel(P1 and P2) = sel(P1) * sel(P2) } \)
- INDEPENDENCE ASSUMPTION

disjunction:
- \( \text{sel( grade = 'C' or course = '415') } \)
- \( \text{sel(P1 or P2) = sel(P1) + sel(P2) - sel(P1 and P2) } \)
- \( = \text{sel(P1) + sel(P2) - sel(P1) * sel(P2) } \)
- INDEPENDENCE ASSUMPTION, again

disjunction: in general
\[
\text{sel(P1 or P2 or ... Pn) =}
1 - (1 - \text{sel(P1) } ) * (1 - \text{sel(P2) } ) * ... (1 - \text{sel(Pn) })
\]
Selections – summary

- $\text{sel}(A=\text{constant}) = \frac{1}{V(A)}$
- $\text{sel}(A>a) = \frac{(A_{\text{max}} - a)}{(A_{\text{max}} - A_{\text{min}})}$
- $\text{sel}(\neg P) = 1 - \text{sel}(P)$
- $\text{sel}(P_1 \text{ and } P_2) = \text{sel}(P_1) \times \text{sel}(P_2)$
- $\text{sel}(P_1 \text{ or } ... \text{ or } P_n) = 1 - (1 - \text{sel}(P_1)) \times ... \times (1 - \text{sel}(P_n))$

- 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 $R_{\text{cols}} \cap S_{\text{cols}} = \emptyset$?
  - $R_{\text{cols}} \cap S_{\text{cols}}$ is a key for R (and a Foreign Key in S)?

\[ nr \times ns \]

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 $R_{\text{cols}} \cap S_{\text{cols}} = \emptyset$? \hspace{10pt} \textbf{nr * ns}
  - $R_{\text{cols}} \cap S_{\text{cols}}$ is a key for R (and a Foreign Key in S)?
Results 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)?

\[
\begin{align*}
\text{nr} & \quad \downarrow \\
\text{ns} & \quad \uparrow
\end{align*}
\]

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

  \[\text{Hint: for a given tuple of R, how many tuples of S will it match?}\]

- match each R-tuple with S-tuples
  \[
  \text{est}_\text{size} < \frac{\text{NTuples(R) \cdot NTuples(S)}}{\text{NKeys(A,S)}} \leq \frac{\text{nr} \cdot \text{ns}}{V(A,S)}
  \]

- symmetrically, for S:
  \[
  \text{est}_\text{size} < \frac{\text{NTuples(R) \cdot NTuples(S)}}{\text{NKeys(A,R)}} \leq \frac{\text{nr} \cdot \text{ns}}{V(A,R)}
  \]

- Overall:
  \[
  \text{est}_\text{size} \approx \frac{\text{NTuples(R)} \cdot \text{NTuples(S)}}{\text{MAX}[\text{NKeys(A,S)}, \text{NKeys(A,R)}]}
  \]
On the Uniform Distribution Assumption

- Assuming uniform distribution is rather crude

Histograms

- For better estimation, use a histogram

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
plan generation

- Selections – eg.,
  \[
  \text{select } * \\
  \text{from TAKES} \\
  \text{where grade = 'A'}
  \]

- Plans?

Cost estimation

REMINDER

<table>
<thead>
<tr>
<th>Scan</th>
<th>Eq</th>
<th>Range</th>
<th>Ins</th>
<th>Del</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>B</td>
<td>B/2</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>sorted</td>
<td>B</td>
<td>logB</td>
<td>&lt;= +m</td>
<td>Search+1</td>
</tr>
<tr>
<td>Cluster</td>
<td>1.5B</td>
<td>h</td>
<td>&lt;= -m</td>
<td>Search+3</td>
</tr>
<tr>
<td>u-tree</td>
<td>~B</td>
<td>1+h'</td>
<td>&lt;= +m'</td>
<td>Search+2</td>
</tr>
<tr>
<td>u-hash</td>
<td>~B</td>
<td>~2</td>
<td>B</td>
<td>Search+2</td>
</tr>
</tbody>
</table>

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)

\[
\begin{array}{c}
\text{Sr} \\
\text{fr} \\
#1 \\
#2 \\
\ldots \\
#br
\end{array}
\]

plan generation

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

\[
\begin{array}{c}
\text{Sr} \\
\text{fr} \\
#1 \\
#2 \\
\ldots \\
#br
\end{array}
\]

plan generation

estimation of selection cardinalities \( SC(A,r) \): non-trivial – we saw it earlier

\[
\begin{array}{c}
\text{Sr} \\
\text{fr} \\
#1 \\
#2 \\
\ldots \\
#br
\end{array}
\]
plan generation

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

case#1: primary key

case#2: sec. key – clustering index

HTi + SC(A,e)/fr

fr
Sr
#1
#2
... #br

case#3: sec. key – non-clust. index

HTi

HTi + 1

#1
#2
... #br

...
plan generation

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

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

Cardena’s formula

- q: # qual records
- Q: # qual. blocks
- N: # records total
- B: # blocks total
- Q=??

Alfonso Cardenas
(IBM->UCLA)
Cardena’s formula

• Pessimistic:
  – \( Q = q \)

• More realistic
  – \( Q = q \) if \( q \leq B \)
  – \( Q = B \) otherwise

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

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

- Prob (single shot, hits our favorite block)
- Prob (it avoids it)
- Prob (it avoids it, q times)
Cardena’s formula

- Cardenas’ formula

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

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

---

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>
</table>

Statistics for Optimization

- NCARD (T) - cardinality of relation T in tuples
- TCARD (T) - number of pages containing tuples from T
- P(T) = TCARD(T)/(# 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>expr</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr = value</td>
<td>$F = \frac{1}{\text{ICARD}(\text{attr index})}$ – if index exists</td>
</tr>
<tr>
<td></td>
<td>$F = \frac{1}{10}$ otherwise</td>
</tr>
<tr>
<td>attr1 = attr2</td>
<td>$F = \frac{\max(\text{ICARD}(I1),\text{ICARD}(I2))}{\text{ICARD}(I1)}$</td>
</tr>
<tr>
<td></td>
<td>$F = \frac{1}{\text{ICARD}(I1)}$ – if only index $i$ exists, or</td>
</tr>
<tr>
<td></td>
<td>$F = \frac{1}{10}$ otherwise</td>
</tr>
<tr>
<td>val1 &lt; attr &lt; val2</td>
<td>$F = \frac{\text{value2} - \text{value1}}{\text{high key} - \text{low key}}$</td>
</tr>
<tr>
<td></td>
<td>$F = \frac{1}{4}$ otherwise</td>
</tr>
<tr>
<td>expr1 or expr2</td>
<td>$F = F(\text{expr1}) + F(\text{expr2}) - F(\text{expr1})F(\text{expr2})$</td>
</tr>
<tr>
<td>expr1 and expr2</td>
<td>$F = F(\text{expr1}) \cdot F(\text{expr2})$</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>Access Path Case</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique index matching equal predicate</td>
<td>$1 + \frac{1}{W}$</td>
</tr>
<tr>
<td>Clustered index $I$ matching $\geq 1$ preds</td>
<td>$F(\text{preds}) \cdot (\text{NINDX}(I) + \text{TCARD}) + W \cdot \text{RSICARD}$</td>
</tr>
<tr>
<td>Non-clustered index $I$ matching $\geq 1$ preds</td>
<td>$F(\text{preds}) \cdot (\text{NINDX}(I) + \text{NCARD}) + W \cdot \text{RSICARD}$</td>
</tr>
<tr>
<td>Segment scan</td>
<td>$\frac{\text{TCARD}}{P} + W \cdot \text{RSICARD}$</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

• \( r_1 \) JOIN \( r_2 \) JOIN ... 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 \( \Rightarrow \) 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 alternative plans
  - Single relation
  - Multiple relations
    - Main idea
    - Dynamic programming – reminder
    - Example
- Estimate cost; pick best

(Reminder: Dynamic Programming)

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:

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

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

So, best price is $1,500 – which legs?
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 \( R \join S \join T \)
Q-opt and Dyn. Programming

- 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
    ```
    select *
    from R, S, T
    where R.a = S.a and S.b = T.b
    order by R.a
    ```

Q-opt and Dyn. Programming

- Details: how to record the fact that, say R is sorted on R.a? or that the user requires sorted output?
- A: record orderings, in the state
- E.g., consider the query
  ```
  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

```
R join S
    T
150 (SM)
R
S
T
2,500 (NL)
```

R join S join T...
Q-opt and Dyn. Programming

• E.g., compute \( R \) join \( S \) join \( T \) order by \( R.a \)

Any other changes?

Q-opt and Dyn. Programming

• 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
1. Enumerate relation orderings:

Prune plans with cross-products immediately!

2. Enumerate join algorithm choices:

+ do same for 4 other plans

→ 4³ = 16 plans so far.

3. Enumerate access method choices:

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

Example:

![Query Plan Diagram]

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

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

Equivalent non-nested query:

```
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 (ie., no joins): look at stats
- n-way joins: left-deep join trees; ie., 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