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

```
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

Canonical form

STUDENT | TAKES

Q-opt - example

STUDENT | TAKES

Hash join; merge join; nested loops;
Index; seq scan
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?

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

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

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

Equivalence of expressions

\[ \sigma_p(R1 \cup R2) \]
\[ t \in LHS \iff \]
\[ t \in (R1 \cup R2) \land P(t) \iff \]
\[ (t \in R1 \lor t \in R2) \land P(t) \iff \]
\[ (t \in R1 \land P(t)) \lor (t \in R2 \land P(t)) \iff \]
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??

\[ \pi_{s}(R1 \setminus R2) \Rightarrow \pi_{s}(R1) \setminus \pi_{s}(R2) \]

Equivalence of expressions

- Selections
  - perform them early
  - break a complex predicate, and push
    \[ \sigma_{p_{1} \land \ldots \land p_{m}}(R) = \sigma_{p_{1}}(\sigma_{p_{2}}(\ldots \sigma_{p_{m}}(R))\ldots) \]
  - simplify a complex predicate
    - \((X=Y \text{ and } Y=3) \Rightarrow \text{\textquotesingle}X=3 \text{ and } Y=3\text{\textquotesingle}\)
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
  \[ R \bowtie S = S \bowtie R \]
  \[ (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \]
  – 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

- Queries
  - `Select * From Blah B Where B.blah = blah`
- 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

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’:
  – fi: average fanout (~50-100)
  – HTi: # levels of index ‘i’ (~2-3)
    • ~ log(#entries)/log(fi)
  – 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 (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 $\text{sel}(P)$ of predicate $P$:
  - fraction of tuples that qualify
  - $\text{sel}(P) = \frac{SC(P)}{nr}$

- eg., assume that $V(\text{grade}, \text{TAKES})=5$ distinct values

- simple predicate $P$: $A=\text{constant}$
  - $\text{sel}(A=\text{constant}) = \frac{1}{V(A,r)}$
  - eg., $\text{sel}(\text{grade}='B') = \frac{1}{5}$
  - (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 = '415')
  – sel(P1 and P2) = sel(P1) * sel(P2)
  – INDEPENDENCE ASSUMPTION


Selections – complex predicates

disjunction:
  – sel( grade = 'C' or course = '415')
  – sel(P1 or P2) = sel(P1) + sel(P2) – sel(P1 and P2)
  – = sel(P1) + sel(P2) – sel(P1)*sel(P2)
  – INDEPENDENCE ASSUMPTION, again
Selections – complex predicates

disjunction: in general
\[ sel(P_1 \text{ or } P_2 \text{ or } \ldots \text{ or } P_n) = 1 - (1 - sel(P_1)) \times (1 - sel(P_2)) \times \ldots \times (1 - sel(P_n)) \]

Selections – summary

- \( sel(A=\text{constant}) = 1/V(A,r) \)
- \( sel(A>a) = (A_{\text{max}} - a) / (A_{\text{max}} - A_{\text{min}}) \)
- \( sel(\neg P) = 1 - sel(P) \)
- \( sel(P_1 \text{ and } P_2) = sel(P_1) \times sel(P_2) \)
- \( sel(P_1 \text{ or } P_2) = sel(P_1) + sel(P_2) - sel(P_1) \times sel(P_2) \)
- \( sel(P_1 \text{ or } \ldots \text{ or } P_n) = 1 - (1-sel(P_1)) \times \ldots \times (1-sel(P_n)) \)

- 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)?
Result Size Estimation for Joins

- General case: \( R_{\text{cols}} \cap S_{\text{cols}} = \{A\} \) (and \( A \) is key for neither)
  - match each \( R \)-tuple with \( S \)-tuples
    \[
    \text{est}_\text{size} \leftarrow \frac{\text{NTuples}(R) \times \text{NTuples}(S)}{\text{NKeys}(A,S)} \leftarrow \frac{nr \times ns}{V(A,S)}
    \]
  - symmetrically, for \( S \):
    \[
    \text{est}_\text{size} \leftarrow \frac{\text{NTuples}(R) \times \text{NTuples}(S)}{\text{NKeys}(A,R)} \leftarrow \frac{nr \times ns}{V(A,R)}
    \]
  - Overall:
    \[
    \text{est}_\text{size} = \frac{\text{NTuples}(R) \times \text{NTuples}(S)}{\text{MAX}\{\text{NKeys}(A,S), \text{NKeys}(A,R)\}}
    \]

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 (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?

<table>
<thead>
<tr>
<th>Sr</th>
<th>fr</th>
<th>#1</th>
<th>#2</th>
<th>#br</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

seq. scan – cost?
- $br$ (worst case)
- $\frac{br}{2}$ (average, if we search for primary key)

plan generation

binary search – cost?
if sorted and consecutive:
- $\sim \log(br) +$
- $\frac{SC(A,r)}{fr}$ (=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?
- levels of index +
- blocks w/ qual. tuples

case#1: primary key

case#2: sec. key – clustering index

HTi + SC(A,r)/fr

case#3: sec. key – non-clust. index

HTi + 1
plan generation

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

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

Cardena’s formula

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

Alfonso Cardenas
(UCLA)
Cardena’s formula

- **Pessimistic:**
  - \( Q = q \)

- **More realistic**
  - \( Q = q \) if \( q \leq B \)
  - \( Q = B \) otherwise

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

Citation

Access path selection in a relational
database management system. In SIGMOD

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

Costs per Access Path Case

<table>
<thead>
<tr>
<th>Case</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})<em>(\text{NINDX(I)}+\text{TCARD})+W</em>\text{RSICARD} )</td>
</tr>
<tr>
<td>Non-clustered index I matching &gt;=1 preds</td>
<td>( F(\text{preds})<em>(\text{NINDX(I)}+\text{NCARD})+W</em>\text{RSICARD} )</td>
</tr>
<tr>
<td>Segment scan</td>
<td>( \text{TCARD/P} + W*\text{RSICARD} )</td>
</tr>
</tbody>
</table>
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

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 ⇒ 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
(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:
So, best price is $1,500 – which legs?

A: follow the winning edges, backwards
(Reminder: Dynamic Programming)

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 Dynamic 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
```

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

```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 \texttt{R join S join T order by R.a}

\begin{itemize}
  \item 150 (SM)
  \item 2,500 (NL)
  \item \texttt{R join S}
  \item \texttt{R join S join T}
  \item \texttt{R join S join T}, sorted \texttt{R.a}
\end{itemize}

Any other changes?

\begin{itemize}
  \item \texttt{R join S (R.a)}
  \item \texttt{R join S join T, sorted R.a}
  \item \texttt{R join S join T}
\end{itemize}
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 of relation orderings]

   Prune plans with cross-products immediately!

2. Enumerate join algorithm choices:

   ![Diagram of join algorithm choices]

   + do same for 4 other plans
   \( \Rightarrow 4^4 = 16 \) 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:

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
  – 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

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

Structure of query optimizers:

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 (e.g., 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