L12: Query Optimization

Note: Slides whose titles are put in () are for your reference only. Details will be covered in COMP9315.

Query Optimization
Query Optimization

SQL query

parse

parse tree

convert

logical query plan

apply laws

“improved” l.q.p

estimate result sizes

l.q.p. + sizes

consider physical plans

heuristic-based q.o.

{P1,P2,…..}

{(P1,C1),(P2,C2)…}

plan

its cost

execute

answer

pick best

estimate costs

statistics

cost-based q.o.

DBMS-L12: Query Optimization -- 2
Overview of Query Optimization

- Plan: Tree of R.A. ops, with choice of alg for each op.
  - Each operator typically implemented using a `pull' interface: when an operator is `pulled' for the next output tuples, it `pulls' on its inputs and computes them.

- Two main issues:
  - For a given query, what plans are considered?
    - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?

- Ideally: Want to find best plan. Practically: Avoid worst plans!
Example

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND R.bid=100 AND S.rating>5
```

RA Tree:

```
\( \pi_{\text{sname}} (\sigma_{\text{bid}=100 \land \text{rating}>5} (\text{Reserves} \bowtie\bowtie \text{Sailor})) \)
```

```
\( \pi_{\text{sname}} (\sigma_{\text{bid}=100} (\text{Reserves} \bowtie\bowtie \sigma_{\text{rating}>5} (\text{Sailor}))) \)
```

… (many more)
Schema for Examples

Sailors ($sid$: integer, $sname$: string, $rating$: integer, $age$: real)
Reserves ($sid$: integer, $bid$: integer, $day$: dates, $rname$: string)

- Similar to old schema; $rname$ added for variations.
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
Motivating Example

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
    R.bid=100 AND S.rating > 5
```

- B = 5
- Cost: \(1000 + \text{ceil}(1000/3) \times 500 = 168,000\) I/Os
- By no means the worst plan!
- Misses several opportunities: wrong choice of outer relation, selections could have been `pushed’ earlier, no use is made of any available indexes, etc.
- **Goal of optimization:** To find more efficient plans that compute the same answer.
Alternative Plans 1 (No Indexes)

- Main difference: push selects.
- With 5 buffers, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, assume we have 100 boats + uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, assume we have 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*4*250), merge (10+250). Note: $5 < \sqrt{250}$
  - Total: 4060 page I/Os.

- If we used BNL join, join cost = 10+4*250, total cost = 2770.
- If we `push' projections, T1 has only sid, T2 only sid and sname:
  - T1 fits in 3 pages, cost of BNL drops to under 250 pages,
  - total < 2000.
Alternative Plans 2 With Indexes

- **With clustered index on bid of Reserves**, we get $\frac{100,000}{100} = 1000$ tuples on $\frac{1000}{100} = 10$ pages.

- INL with pipelining (outer is not materialized)
  - Projecting out unnecessary fields from outer doesn’t help.

- Join column sid is a key for Sailors.
  - At most one matching tuple, unclustered hash index on sid OK.

- Decision not to push rating>5 before the join is based on availability of sid index on Sailors.

- Cost: Selection of Reserves tuples (10 I/Os); for each, must get matching Sailors tuple ($1000 \times (1.2+1)$); total 1210 I/Os.
Heuristic-based Query Optimization

- Relies on relational algebra equivalence
- 3 major heuristics
  - Push down selection
  - Push down projection
  - Combine Cartesian product and selection into join
Relational Algebra Equivalences

- Allow us to choose different join orders and to `push’ selections and projections ahead of joins.

- Selections: $\sigma_{c_1 \land \ldots \land c_n}(R) \equiv \sigma_{c_1}(\ldots \sigma_{c_n}(R))$  
  (Cascade) $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$  
  (Commute)

- Projections: $\pi_{a_1}(R) \equiv \pi_{a_1}(\ldots(\pi_{a_n}(R)))$  
  (Cascade)

- Joins: $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$  
  (Associative)  
  $(R \bowtie S) \equiv (S \bowtie R)$  
  (Commute)

*Note: Conditions apply. See textbooks for the complete equivalences*
More Equivalences (Interactions)

- A projection commutes with a selection that only uses attributes retained by the projection.
- Selection between attributes of the two arguments of a cross-product converts cross-product to a join.
- A selection on just attributes of R commutes with $R \bowtie S$. (i.e., $\sigma (R \bowtie S) \equiv \sigma (R) \bowtie S$)
- Similarly, if a projection follows a join $R \bowtie S$, we can `push’ it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.
Heuristic-based Q.O. Example

EMP (ssn, fname, lname, bdate)
PROJ (pno, pname)
WORKS_ON (ssn, pno)

```
SELECT lname
FROM  EMP E, WORKS_ON W, PROJ P
WHERE P.pname = 'foobar'
  AND P.pno = W.pno
  AND W.ssn = E.ssn
  AND E.bdate > '31/12/1957'
```
Q: What are the algebraic equivalences used in this step?

Heurisic-based Q.O. Example

\[ \pi_{\text{name}} (\sigma_{\text{pname}='\text{foobar'} \land \text{pno}=\text{pno} \land \text{ssn}=\text{ssn} \land \text{bdate}>'31/12/1957')} \]

Push Sel

X

\[ \pi_{\text{name}} \]

\[ (\sigma_{\text{pno}=\text{pno}}) \]

\[ (\sigma_{\text{ssn}=\text{ssn}}) \]

\[ (\sigma_{\text{bdate}>'31/12/1957}') \]

EMP

WORKS_ON

PROJ

Query Optimization

EMP

WORKS_ON

PROJ

DBMS-L12: Query Optimization -- 13
### Heuristic-based Q.O. Example

Change join Order (most selective done first)

Q: What are the algebraic equivalences used in this step?
Q: What are the algebraic equivalences used in this step?

Heuristic-based Q.O. Example

Use joins

Query Optimization
Q: What are the algebraic equivalences used in this step?

Heuristics-based Q.O. Example

Push projection

\[ \pi_{\text{name}} \]
\[ \sigma_{\text{bdate} > '31/12/1957'} \]
\[ \pi_{\text{ssn}} \]
\[ \sigma_{\text{ pname = 'foobar'}} \]
\[ \pi_{\text{ssn, pno}} \]
\[ \pi_{\text{ssn}} \]
\[ \pi_{\text{pno}} \]
\[ \pi_{\text{pno}} \]
\[ \sigma_{\text{bdate} > '31/12/1957'} \]
\[ \pi_{\text{ssn, pno}} \]
\[ \pi_{\text{ssn}} \]
\[ \pi_{\text{pno}} \]
\[ \pi_{\text{ssn}} \]
\[ \pi_{\text{pno}} \]
\[ \pi_{\text{ssn, pno}} \]

EMP

WORKS_ON

PROJ
Heuristic-based Q.O. Problems

No heuristics is *always* good.

- “goodness”: eliminate catesian product > early selection > early projection

Q: Can you find examples in which other heuristics are not good?

What if we have index on PROJ.pno and WORKS_ON is very small?
Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate cost of each operation in plan tree.
    - Depends on input cardinalities.
    - We’ve already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must estimate size of result for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.

- We’ll discuss the System R cost estimation approach.
  - Very inexact, but works ok in practice.
  - More sophisticated techniques known now.
Statistics and Catalogs

- Need information about the relations and indexes involved. Catalogs typically contain at least:
  - # tuples (NTuples) and # pages (NPages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.

- Catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.

- More detailed information (e.g., histograms of the values in some field) are sometimes stored.
Size Estimation and Reduction Factors

Consider a query block:

- Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- Reduction factor (RF) associated with each term reflects the impact of the term in reducing result size. Result cardinality = Max # tuples * product of all RF’s.

- Implicit assumption that terms are independent!
- Term col=value has RF 1/NKeys(I), given index I on col
- Term col1=col2 has RF 1/MAX(NKeys(I1), NKeys(I2))
- Term col>value has RF (High(I)-value)/(High(I)-Low(I))
Enumeration of Alternative Plans

- There are two main cases:
  - Single-relation plans
  - Multiple-relation plans

- For queries over a single relation, queries consist of a combination of selects, projects, and aggregate ops:
  - Each available access path (file scan / index) is considered, and the one with the least estimated cost is chosen.
  - The different operations are essentially carried out together (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are pipelined into the aggregate computation).
Cost Estimates for Single-Relation Plans

- **Index I on primary key matches selection:**
  - Cost is $\text{Height}(I)+1$ for a B+ tree, about 1.2 for hash index.

- **Clustered index I matching one or more selects:**
  - $(\text{NPages}(I)+\text{NPages}(R)) \times \text{product of RF’s of matching selects}$. 

- **Non-clustered index I matching one or more selects:**
  - $(\text{NPages}(I)+\text{NTuples}(R)) \times \text{product of RF’s of matching selects}$. 

- **Sequential scan of file** (always possible):
  - $\text{NPages}(R)$.

- **Note:** Typically, no duplicate elimination on projections! (Exception: Done on answers if user says DISTINCT.)
Example

If we have an index on rating:
- \((1/N\text{Keys}(I)) \times NTuples(R) = (1/10) \times 40000\) tuples retrieved.
- Clustered index: \((1/N\text{Keys}(I)) \times (N\text{Pages}(I)+N\text{Pages}(R)) = (1/10) \times (50+500)\) pages are retrieved. (This is the cost.)
- Unclustered index: \((1/N\text{Keys}(I)) \times (N\text{Pages}(I)+NTuples(R)) = (1/10) \times (50+40000)\) pages are retrieved.

If we have an index on sid (non-matching index):
- Would have to retrieve all tuples/pages. With a clustered index, the cost is 50+500, with unclustered index, 50+40000.
- More expensive than a file scan

Doing a file scan (also known as table(space) scan):
- We retrieve all file pages (500).

Other plans omitted in this course (e.g., index-only access, if having index on <rating, sid>)

Q: What other information do you need in order to calculate exact cost for the clustered index case?

Assume: \(N\text{Pages}(I) = 50\); No opt. for unclustered index access is used.

```
SELECT S.sid
FROM Sailors S
WHERE S.rating=8
```
Queries Over Multiple Relations

- Fundamental decision in System R: only left-deep join trees are considered.
  - As the number of joins increases, the number of alternative plans grows rapidly; we need to restrict the search space.
  - Left-deep trees allow us to generate all fully pipelined plans.
    - Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).
    - E.g., for 3 relations (A join B join C), left-deep trees are ((A,B),C), ((B,A),C), ((B,C),A) and ((C,B),A) if no cartesian product is allowed.
Highlights of System R Optimizer

- **Impact:**
  - Most widely used currently; works well for < 10 joins.

- **Cost estimation:** Approximate art at best.
  - Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes.
  - Considers combination of CPU and I/O costs.

- **Plan Space:** Too large, must be pruned.
  - Only the space of left-deep plans is considered.
    - Left-deep plans allow output of each operator to be pipelined into the next operator without storing it in a temporary relation.
  - Cartesian products avoided.
Enumeration of Left-Deep Plans

- Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.

- Enumerated using N passes (if N relations joined):
  - [dynamic programming paradigm]
    - Pass 1: Find best 1-relation plan for each relation.
    - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. (All 2-relation plans.)
    - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N’th relation. (All N-relation plans.)

- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each interesting order of the tuples.
(Enumeration of Plans (Contd.))

- ORDER BY, GROUP BY, aggregates etc. handled as a final step, using either an `interestingly ordered’ plan or an additional sorting operator.
- An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
  - i.e., avoid Cartesian products if possible.
- In spite of pruning plan space, this approach is still exponential in the # of tables.
(Example)

- Pass 1:
  - **Sailors**: B+ tree matches rating > 5, and is probably cheapest. However, if this selection is expected to retrieve a lot of tuples, and index is unclustered, file scan may be cheaper.
    - Still, B+ tree plan kept (because tuples are in rating order).
  - **Reserves**: B+ tree on bid matches bid = 500; cheapest.

- Pass 2:
  - We consider each plan retained from Pass 1 as the outer, and consider how to join it with the (only) other relation.
    - e.g., Reserves as outer: Hash index can be used to get Sailors tuples that satisfy sid = outer tuple’s sid value.
Summary

- Query optimization is an important task in a relational DBMS.
- Must understand optimization in order to understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
- Two parts to optimizing a query:
  - Consider a set of alternative plans.
    - Must prune search space; typically, left-deep plans only.
  - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - Key issues: Statistics, indexes, operator implementations.
Summary (Contd.)

- **Single-relation queries:**
  - All access paths considered, cheapest is chosen.
  - Issues: Selections that match index, whether index key has all needed fields and/or provides tuples in a desired order.

- **Multiple-relation queries:**
  - Need to consider all possible plans and pick the one with the lowest (estimated) cost.
  - **System R (Selinger-style optimizer)**
    - All single-relation plans are first enumerated.
      - Selections/projections considered as early as possible.
    - Next, for each 1-relation plan, all ways of joining another relation (as inner) are considered.
    - Next, for each 2-relation plan that is `retained`, all ways of joining another relation (as inner) are considered, etc.
    - At each level, for each subset of relations, only best plan for each interesting order of tuples is `retained`.

In this course, we only deal with up to 2 joins (3 base relations). Thus, can enumerate all alternative plans and select the least costly one. See tutorials/reviews for examples.
Supplementary Example

SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND R.bid=100 AND S.rating>5

| R | = 100k, ||R|| = 1000
| S | = 40k, ||S|| = 500
100 different bid
10 different rating

Clustering I: R.bid, S.rating, R.sid
Unclustering I: S.sid

1. Choose Join order (left-deep)
2. Choose Join alg & selection position & pipeline/bloected-mode

- Branch-and-bound:
  - Calculate the cost of one seemingly-good plan. Let it be X
  - Consider alternative plans, discard it immediately if the current cost is >X already.

Query Optimization
How to choose a good plan?

- Rules of thumb:
  - Try INL when out rel has a small cardinality
  - Try HJ, then SMJ, then BNL with the increase number of buffer pages

Clustered I: R.bid, S.rating, R.sid

Unclustered I: S.sid

1. Choose Join order (left-deep)
2. Choose Join alg & selection position & pipeline/bloced-mode

- Branch-and-bound:
  - Calculate the cost of one seemingly-good plan. Let it be X
  - Consider alternative plans, discard it immediately if the current cost is >X already.