QUERY OPTIMIZATION

CS 564- Fall 2015

ACKs: Jeff Naughton, Jignesh Patel, AnHai Doan
EXAMPLE QUERY

• EMP(ssn, ename, addr, sal, did)
  – 10000 tuples, 1000 pages
• DEPT(did, dname, floor, mgr)
  – 500 tuples, 50 pages

SELECT  DISTINCT  ename 
FROM       Emp  E, Dept  D  
WHERE        E.did  =  D.did  
AND            D.dname  =  'Toy'  ;
**EVALUATION PLAN (1)**

```sql
SELECT DISTINCT ename
FROM Emp E, Dept D
WHERE E.did = D.did
AND D.dname = 'Toy';
```
### Evaluation Plan (2)

```
SELECT DISTINCT ename
FROM  Emp E, Dept D
WHERE E.did = D.did
AND   D.dname = 'Toy';
```
SELECT DISTINCT ename
FROM Emp E, Dept D
WHERE E.did = D.did
AND D.dname = 'Toy';

buffer size B= 50
Pipelined Evaluation

Instead of materializing the temporary relation to disk, we can instead pipeline to the next operator.

• How much cost does it save?
• When can pipelining be used?
**SELECT DISTINCT** `ename` 
**FROM** Emp E, Dept D 
**WHERE** E.did = D.did 
**AND** D.dname = 'Toy';
Query Optimization

- identify candidate equivalent trees
- for each candidate find best annotated version (using available indexes)
- choose the best overall by estimating the cost of each plan

In practice we choose from a subset of possible plans
ARCHITECTURE OF AN OPTIMIZER

query

parsed query

Query Optimizer
- Plan generator
- Plan cost estimator

System Catalog

evaluation plan
Query Optimization

- Plan: Annotated RA Tree
  - operator interface: open/getNext/close
  - pipelined or materialized
- Two main issues:
  - What plans are considered?
  - How is the cost of a plan estimated?
- Ideally: best plan!
- Practically: avoid worst plans! Look at a subset of all plans
**Cost Estimation**

- Estimate the *cost* of each operation in the plan
  - depends on input cardinalities
  - algorithm cost (we know this!)
- Estimate the *size* of result
  - statistics about input relations
  - for selections and joins, we assume independence of predicates
Cost Estimation

- Statistics stored in the catalogs
  - cardinality
  - size in pages
  - # distinct keys (for index)
  - range (for numeric values)
- Catalogs update periodically
- Commercial systems use histograms, which give more accurate estimates
There is a huge space of plans to navigate through

- Relational algebra *equivalences* help to construct many alternative plans
**Equivalence (1)**

- **Commutativity** of $\sigma$
  \[
  \sigma_{P_1} (\sigma_{P_2} (R)) \equiv \sigma_{P_2} (\sigma_{P_1} (R))
  \]

- **Cascading** of $\sigma$
  \[
  \sigma_{P_1 \land P_2 \land \ldots \land P_n} (R) \equiv \sigma_{P_1} (\sigma_{P_2} (\ldots \sigma_{P_n} (R) \ldots))
  \]

- **Cascading** of $\pi$
  \[
  \pi_{\alpha_1} (R) \equiv \pi_{\alpha_1} (\pi_{\alpha_2} (\ldots \pi_{\alpha_n} (R) \ldots)) \text{ when } \alpha_i \subseteq \alpha_{i+1}
  \]

This means that we can evaluate selections in any order!
**Equivalence (2)**

- **Commutativity** of join
  \[ R \bowtie S \equiv S \bowtie R \]

- **Associativity** of join
  \[ (R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T) \]

This means that we can reorder the computation of joins in any way!
EQUIVALENCE (3)

• Selections + Projections

\[
\sigma_P (\pi_a (R)) \equiv \pi_a (\sigma_P (R)) \quad \text{(if the selection involves attributes that remain after projection)}
\]

• Selections + Joins

\[
\sigma_P (R \bowtie S) \equiv \sigma_P (R) \bowtie S \quad \text{(if the selection involves attributes only in S)}
\]

This means that we can push selections down the plan tree!
EVALUATION PLANS

Single relation plan (no joins)

• file scan
• index scan(s): clustered or non-clustered
  – More than one index may “match” predicates
• Choose the one with the least estimated cost
• Merge/pipeline selection and projection (and aggregate)
  – Index aggregate evaluation
EVALUATION PLANS

Multiple relation plan

• selections can be combined into joins
• joins can be reordered
• selections and projections can be pushed down the plan tree
JOIN REORDERING

Consider the following join: \( R \bowtie S \bowtie T \bowtie U \)

- Left-deep join plans
- Allow for fully pipelined evaluation