Query Evaluation

• **Problem:** An SQL query is declarative - does not specify a query execution plan.
  
  • A relational algebra expression is procedural - there is an associated query execution plan.

• **Solution:** Convert SQL query to an equivalent relational algebra and evaluate it using the associated query execution plan.

  – But which equivalent expression is best?
Naive Conversion

SELECT DISTINCT TargetList
FROM R1, R2, ..., RN
WHERE Condition

is equivalent to

\[ \pi_{TargetList} (\sigma_{Condition} (R1 \times R2 \times ... \times RN)) \]

but this may imply a very inefficient query execution plan.

Example: \[ \pi_{Name} (\sigma_{Id=ProfId \land CrsCode='CS532'} (Professor \times Teaching)) \]
  - Result can be < 100 bytes
  - But if each relation is 50K then we end up computing an intermediate result Professor \times Teaching of size 2500M before shrinking it down to just a few bytes.

Problem: Find an equivalent relational algebra expression that can be evaluated “efficiently”.
Query Processing Architecture

1. SQL Query
2. SQL Parser
3. Relational Algebra Expression
4. Query Optimizer
   - Query Plan Generator
   - Cost Estimator
5. System Catalog
6. Query Execution Plan
7. Query Plan Interpreter
8. Query Result
Query Optimizer

• Uses heuristic algorithms to evaluate relational algebra expressions. This involves:
  – estimating the cost of a relational algebra expression
  – transforming one relational algebra expression to an equivalent one
  – choosing access paths for evaluating the subexpressions

• Query optimizers do not “optimize” - just try to find “reasonably good” evaluation strategies
Equivalence Preserving Transformations

• To transform a relational expression into another equivalent expression we need transformation rules that preserve equivalence

• Each transformation rule
  – Is provably correct (ie, does preserve equivalence)
  – Has a heuristic associated with it
Selection and Projection Rules

• Break complex selection into simpler ones:
  \[ \sigma_{\text{Cond}_1 \land \text{Cond}_2} (R) \equiv \sigma_{\text{Cond}_1} (\sigma_{\text{Cond}_2} (R)) \]

• Break projection into stages:
  \[ \pi_{\text{attr}} (R) \equiv \pi_{\text{attr}} (\pi_{\text{attr}'} (R)), \text{ if } \text{attr} \subseteq \text{attr}' \]

• Commute projection and selection:
  \[ \pi_{\text{attr}} (\sigma_{\text{Cond}} (R)) \equiv \sigma_{\text{Cond}} (\pi_{\text{attr}} (R)), \text{ if } \text{attr} \supseteq \text{all attributes in Cond} \]
Commutativity and Associativity of Join
(and Cartesian Product as Special Case)

- **Join commutativity:** \( R \bowtie S \equiv S \bowtie R \)
  - used to reduce cost of nested loop evaluation strategies (smaller relation should be in outer loop)

- **Join associativity:** \( R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \)
  - used to reduce the size of intermediate relations in computation of multi-relational join - first compute the join that yields smaller intermediate result

- **N-way join** has \( T(N) \times N! \) different evaluation plans
  - \( T(N) \) is the number of parenthesized expressions
  - \( N! \) is the number of permutations

- Query optimizer **cannot** look at all plans (might take longer to find an optimal plan than to compute query brute-force). Hence it does not necessarily produce optimal plan
Pushing Selections and Projections

- $\sigma_{\text{Cond}}(R \times S) \equiv R \bowtie_{\text{Cond}} S$
  - $\text{Cond}$ relates attributes of both $R$ and $S$
  - Reduces size of intermediate relation since rows can be discarded sooner

- $\sigma_{\text{Cond}}(R \times S) \equiv \sigma_{\text{Cond}}(R) \times S$
  - $\text{Cond}$ involves only the attributes of $R$
  - Reduces size of intermediate relation since rows of $R$ are discarded sooner

- $\pi_{\text{attr}}(R \times S) \equiv \pi_{\text{attr}}(\pi_{\text{attr'}}(R) \times S)$,
  if $\text{attributes}(R) \supseteq \text{attr'} \supseteq \text{attr} \cap \text{attributes}(R)$
  - reduces the size of an operand of product
Equivalence Example

\[ \sigma_{C_1 \land C_2 \land C_3} (R \times S) \equiv \sigma_{C_1} (\sigma_{C_2} (\sigma_{C_3} (R \times S))) \equiv \sigma_{C_1} (\sigma_{C_2} (R) \times \sigma_{C_3} (S)) \equiv \sigma_{C_2} (R) \bowtie_{C_1} \sigma_{C_3} (S) \]

assuming \( C_2 \) involves only attributes of \( R \), \( C_3 \) involves only attributes of \( S \), and \( C_1 \) relates attributes of \( R \) and \( S \)
Cost - Example 1

```
SELECT P.Name
FROM Professor P, Teaching T
WHERE P.Id = T.ProfId             -- join condition
    AND P.DeptId = 'CS' AND T.Semester = 'F1994'

π_{Name}(σ_{DeptId='CS' \land Semester='F1994'}(Professor \bowtie_{Id=ProfId} Teaching))
```

Master query execution plan (nothing pushed)
Metadata on Tables (in system catalogue)

– **Professor** *(Id, Name, DeptId)*
   - *size*: 200 pages, 1000 rows, 50 departments
   - *indexes*: clustered, 2-level B⁺-tree on DeptId, hash on Id

– **Teaching** *(ProflId, CrsCode, Semester)*
   - *size*: 1000 pages, 10,000 rows, 4 semesters
   - *indexes*: clustered, 2-level B⁺-tree on Semester; hash on ProflId

– **Definition:** *Weight of an attribute - average number of rows that have a particular value*
   - *weight of* **Id** = 1 (it is a key)
   - *weight of* **ProflId** = 10 (10,000 classes/1000 professors)
Estimating Cost - Example 1

- Join - block-nested loops with 52 page buffer
  (50 pages - input for Professor, 1 page - input for Teaching
  - Scanning Professor (outer loop): 200 page transfers, (4 iterations, 50 transfers each)
  - Finding matching rows in Teaching (inner loop): 1000 page transfers for each iteration of outer loop
  - Total cost = 200+4*1000 = 4200 page transfers
Estimating Cost - Example 1 (cont’d)

• *Selection* and *projection* - scan rows of intermediate file, discard those that don’t satisfy selection, project on those that do, write result when output buffer is full.

• Complete algorithm:
  – do *join*, write result to intermediate file on disk
  – read intermediate file, do *select/project*, write final result
  – **Problem**: unnecessary I/O
Pipelining

- **Solution:** use *pipelining*:
  - *join* and *select/project* act as coroutines, operate as producer/consumer sharing a buffer in main memory.
    - When *join* fills buffer; *select/project* filters it and outputs result
    - Process is repeated until *select/project* has processed last output from *join*
  - Performing *select/project* adds no additional cost
Estimating Cost - Example 1 (cont’d)

- Total cost:
  \[4200 + \text{(cost of outputting final result)}\]

  - We will *disregard the cost of outputting final result* in comparing with other query evaluation strategies, since this will be same for all
Cost Example 2

\[
\pi_{\text{Name}} \left( \sigma_{\text{Semester}=\text{F1994}} \left( \sigma_{\text{DeptId}=\text{CS}} \left( \text{Professor} \right) \right) \right) \sigma_{\text{Id}=\text{ProfId}} \left( \text{Teaching} \right)
\]

\[
\text{SELECT} \quad \text{P.Name} \\
\text{FROM} \quad \text{Professor P, Teaching T} \\
\text{WHERE} \quad \text{P.Id = T.ProfId AND} \\
\quad \quad \text{P. DeptId = \text{CS AND T.Semester = \text{F1994}}}
\]
Cost Example 2 -- selection

- Compute $\sigma_{DeptId='CS'}$ (Professor) (to reduce size of one join table) using clustered, 2-level B+ tree on DeptId.
  - 50 departments and 1000 professors; hence weight of DeptId is 20 (roughly 20 CS professors).
  These rows are in ~4 consecutive pages in Professor.
  - Cost = 4 (to get rows) + 2 (to search index) = 6
  - keep resulting 4 pages in memory and pipe to next step
Cost Example 2 -- \textit{join}

- Index-nested loops join using hash index on \textit{ProfId} of Teaching and looping on the selected professors (computed on previous slide)
  - Since selection on \textit{Semester} was not pushed, hash index on \textit{ProfId} of Teaching can be used
  - \textit{Note}: if selection on \textit{Semester} were pushed, the index on \textit{ProfId} would have been lost - an advantage of \textit{not} using a fully pushed query execution plan
Cost Example 2 - *join* (cont’d)

– Each professor matches ~10 Teaching rows. Since 20 CS professors, hence 200 teaching records.

– All index entries for a particular *ProfId* are in same bucket. Assume ~1.2 I/Os to get a bucket.

  • Cost = $1.2 \times 20$ (to fetch index entries for 20 CS professors) + 200 (to fetch Teaching rows, since hash index is *unclustered*) = 224
Cost Example 2 - *select/project*

- Pipe result of join to *select* (on *Semester*) and *project* (on *Name*) at no I/O cost
- Cost of output same as for Example 1
- Total cost:
  \[ 6 \text{ (select on Professor)} + 224 \text{ (join)} = 230 \]
- Comparison:
  \[ 4200 \text{ (example 1)} \text{ vs. } 230 \text{ (example 2)} !!! \]
Estimating Output Size

• It is important to estimate the size of the output of a relational expression - size serves as input to the next stage and affects the choice of how the next stage will be evaluated.

• Size estimation uses the following measures on a particular instance of R:
  – $Tuples(R)$: number of tuples
  – $Blocks(R)$: number of blocks
  – $Values(R.A)$: number of distinct values of $A$
  – $MaxVal(R.A)$: maximum value of $A$
  – $MinVal(R.A)$: minimum value of $A$
Estimating Output Size

• For the query:

```
SELECT TargetList
FROM R_1, R_2, ..., R_n
WHERE Condition
```

– Reduction factor is

\[
\frac{\text{Blocks (result set)}}{\text{Blocks}(R_1) \times \ldots \times \text{Blocks}(R_n)}
\]

• Estimates by how much query result is smaller than input
Estimation of Reduction Factor

- Assume that reduction factors due to target list and query condition are independent

- Thus:
  \[ \text{reduction(Query)} = \text{reduction(TargetList)} \times \text{reduction(Condition)} \]
Reduction Due to Simple Condition

- $\text{reduction } (R_i.A=\text{val}) = \frac{1}{\text{Values}(R.A)}$

- $\text{reduction } (R_i.A=R_j.B) = \frac{1}{\max(\text{Values}(R_i.A), \text{Values}(R_j.B))}$

- $\text{reduction } (R_i.A > \text{val}) = \frac{\text{MaxVal}(R_i.A) - \text{val}}{\text{MaxVal}(R_i.A) - \text{MinVal}(R_i.A)}$
Reduction Due to Complex Condition

- \( \text{reduction}(\text{Cond}_1 \text{ AND } \text{Cond}_2) = \text{reduction}(\text{Cond}_1) \times \text{reduction}(\text{Cond}_2) \)
- \( \text{reduction}(\text{Cond}_1 \text{ OR } \text{Cond}_2) = \min(1, \text{reduction}(\text{Cond}_1) + \text{reduction}(\text{Cond}_2)) \)
Reduction Due to TargetList

- $reduction(TargetList) = \frac{\text{number-of-attributes (TargetList)}}{\sum_i \text{number-of-attributes (R}_i)}$
Estimating Weight of Attribute

\[
\text{weight}(R.A) = \text{Tuples}(R) \times \text{reduction}(R.A=value)
\]
Index-Only Queries

• A B⁺ tree index with search key attributes $A_1, A_2, \ldots, A_n$ has stored in it the values of these attributes for each row in the table.
  – Queries involving a prefix of the attribute list $A_1, A_2, \ldots, A_n$ can be satisfied using only the index - no access to the actual table is required.

• Example: Transcript has a clustered B⁺ tree index on StudId. A frequently asked query is one that requests all grades for a given CrsCode.
  – Problem: Already have a clustered index on StudId - cannot create another one (on CrsCode)
  – Solution: Create an unclustered index on (CrsCode, Grade)
    • Keep in mind, however, the overhead in maintaining extra indices
Suppose the following metadata is associated with the Book and Copy tables

**Book**
- 400 pages, 2000 rows
- pages uniformly distributed between 100 and 1000
- hash index on booknum
- clustered 2-level B+tree index on pages

**Copy**
- 1000 pages, 20,000 rows
- price uniformly distributed between 40 and 150
- hash index on booknum
- clustered 2-level B+tree index on price
Practice Problems

Answer the questions based on the following SQL query

```
select title, copynum from Book natural join Copy where price <= 50 and pages >= 991
```

1. Write a relational algebra expression for the query.

2. Create the master execution tree for the relational algebra expression.

3. Create two alternative execution tree for the relational algebra expression.

4. Calculate the number of page reads required for each execution tree. Assume 52 page memory buffer is available.