CSE 544: Relational Operators, Sorting

Wednesday, 5/12/2004

Relational Algebra

- Operates on relations, i.e. *sets*
  - Later: we discuss how to extend this to *bags*
- Five operators:
  - Union: ∪
  - Difference: -
  - Selection: σ
  - Projection: Π
  - Cartesian Product: ×
- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ

1. Union and 2. Difference

- $R_1 \cup R_2$
- Example:

  ActiveEmployees ∪ RetiredEmployees

- $R_1 - R_2$
- Example:

  AllEmployees – RetiredEmployees

What about Intersection ?

- It is a derived operator
- $R_1 \cap R_2 = R_1 - (R_1 - R_2)$
- Also expressed as a join (will see later)
- Example

  UnionizedEmployees ∩ RetiredEmployees

3. Selection

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples

  $\sigma_{Salary > 40000}$ **(Employee)**

  $\sigma_{name = \text{“Smithh”}}$ **(Employee)**
- The condition c can be =, <, ≤, >, ≥, <>

4. Projection

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Example: project social-security number and names:

  $\Pi_{SSN, Name}$ **(Employee)**

  Output schema: **Answer(SSN, Name)**

5. Cartesian Product

- Each tuple in R_1 with each tuple in R_2
- Notation: $R_1 \times R_2$
- Example:

  Employee × Dependents
- Very rare in practice; mainly used to express joins

Cartesian Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

EmployeeSSN	Dname
999999999	Emily
777777777	Joe

Employee x Dependents

Name	SSN	EmployeeSSN	Dname
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Renaming

- Changes the schema, not the instance
- Notation: $\rho_{B1,\ldots,Bn}(R)$
- Example:

  $\rho_{LastName,\ SocSocNo}$ **(Employee)**

  Output schema: **Answer(LastName, SocSocNo)**

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$\rho_{LastName,\ SocSocNo}$ **(Employee)**

LastName	SocSocNo
John	999999999
Tony	777777777

Natural Join

- Notation: $R_1 \,|\times|\, R_2$
- Meaning: $R_1 \,|\times|\, R_2 = \Pi_A(\sigma_C(R_1 \times R_2))$
- Where:
  - The selection σ_C checks equality of all common attributes
  - The projection eliminates the duplicate common attributes

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee $\bowtie$ Dependents =

$\Pi_{Name,\ SSN,\ Dname}(\sigma_{SSN=SSN2}(\text{Employee} \times \rho_{SSN2,\ Dname}(\text{Dependents})))$

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

Natural Join

- R=

A	B
X	Y
X	Z
Y	Z
Z	V

S=

B	C
Z	U
V	W
Z	V

- R |×| S=

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Natural Join

- Given the schemas R(A, B, C, D), S(A, C, E), what is the schema of R |×| S ?
- Given R(A, B, C), S(D, E), what is R |×| S ?
- Given R(A, B), S(A, B), what is R |×| S ?

Theta Join

- A join that involves a predicate
- R1 $|\times|_{\theta}$ R2 = σ_{θ} (R1 × R2)
- Here θ can be any condition: =, <, ≠, ≤, > ≥

Eq-join

- A theta join where θ is an equality
- $R_1 |\times|_{A=B} R_2 = \sigma_{A=B} (R_1 \times R_2)$
- Example:

  Employee $|\times|_{SSN=SSN}$ Dependents

- Most useful join in practice

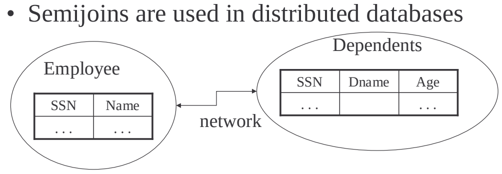
Semijoin

- R |× S = $\Pi_{A1,\ldots,An}$ (R |×| S)
- Where $A_1, \ldots, A_n$ are the attributes in R
- Example:

  Employee |× Dependents

Semijoins in Distributed Databases

- Semijoins are used in distributed databases



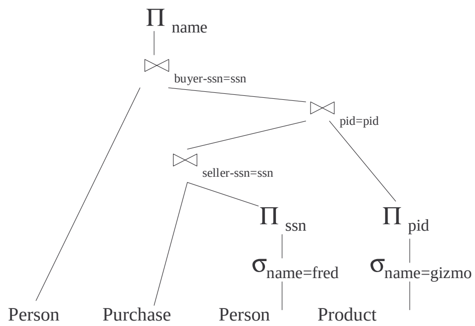
Employee $|\times|_{ssn=ssn}$ ($\sigma_{age>71}$ (Dependents))

T = $\Pi_{SSN}\ \sigma_{age>71}$ (Dependents)

R = Employee |× T

Answer = R |×| Dependents

Complex RA Expressions



Operations on Bags

A **bag** = a set with repeated elements

Relational Engines work on bags, not sets !

All operations need to be defined carefully on bags

- {a,b,b,c}∪{a,b,b,b,e,f,f}={a,a,b,b,b,b,b,c,e,f,f}
- {a,b,b,b,c,c} – {b,c,c,c,d} = {a,b,b,d}
- σ_C(R): preserve the number of occurrences
- Π_A(R): no duplicate elimination
- Cartesian product, join: no duplicate elimination

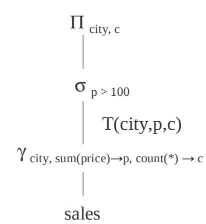
Logical Operators in the Bag Algebra

- Union, intersection, difference
- Selection σ
- Projection Π
- Join ⋈
- Duplicate elimination δ
- Grouping γ
- Sorting τ

(Union through Join: Relational Algebra (on bags))

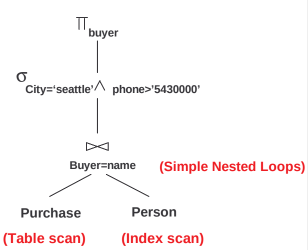
Example

```
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```



Physical Operators

```
SELECT S.buyer
FROM Purchase P, Person Q
WHERE P.buyer=Q.name AND
      Q.city='seattle' AND
      Q.phone > '5430000'
```

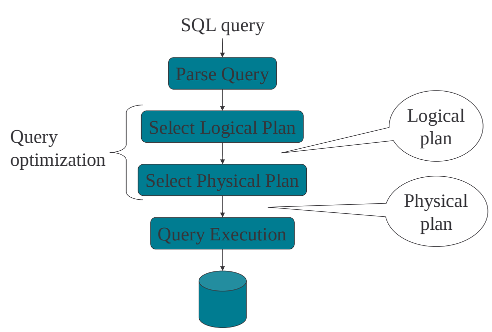


Query Plan:

- logical tree
- implementation choice at every node
- scheduling of operations.

Some operators are from relational algebra, and others (e.g., scan) are not.

Architecture of a Database Engine



Cost Parameters

In database systems the data is on *disks*, not in main memory

The *cost* of an operation = total number of I/Os
Cost parameters:

- B(R) = number of blocks for relation R
- T(R) = number of tuples in relation R
- V(R, a) = number of distinct values of attribute a

Cost Parameters

- *Clustered* table R:
  - Blocks consists only of records from this table
  - B(R) ≈ T(R) / blockSize
- *Unclustered* table R:
  - Its records are placed on blocks with other tables
  - When R is *unclustered*: B(R) ≈ T(R)

- When a is a key, V(R,a) = T(R)
- When a is not a key, V(R,a)

Cost

Cost of an operation =
number of disk I/Os needed to:
- read the operands
- compute the result

Cost of writing the result to disk is *not included* on the following slides

Question: the cost of sorting a table with B blocks ?
Answer:

Scanning Tables

- The table is *clustered*:
  - Table-scan: if we know where the blocks are
  - Index scan: if we have a sparse index to find the blocks
- The table is *unclustered*
  - May need one read for each record

Sorting While Scanning

- Sometimes it is useful to have the output sorted
- Three ways to scan it sorted:
  - If there is a primary or secondary index on it, use it during scan
  - If it fits in memory, sort there
  - If not, use multi-way merge sort

Cost of the Scan Operator

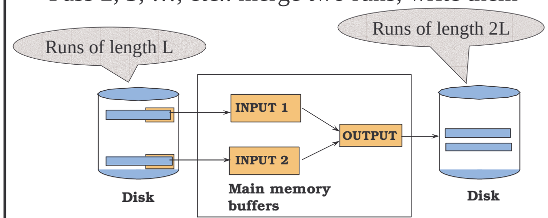
- Clustered relation:
  - Table scan:
    - Unsorted: B(R)
    - Sorted: 3B(R)
  - Index scan
    - Unsorted: B(R)
    - Sorted: B(R) or 3B(R)
- Unclustered relation
  - Unsorted: T(R)
  - Sorted: T(R) + 2B(R)

Sorting

- Problem: sort 1 GB of data with 1MB of RAM.
- Where we need this:
  - Data requested in sorted order (ORDER BY)
  - Needed for grouping operations
  - First step in sort-merge join algorithm
  - Duplicate removal
  - Bulk loading of B+-tree indexes.

2-Way Merge-sort: Requires 3 Buffers in RAM

- Pass 1: Read 1MB, sort it, write it.
- Pass 2, 3, …, etc.: merge two runs, write them



Two-Way External Merge Sort

- Assume block size is B = 4Kb

- Step 1 Ł runs of length L = 1MB
- Step 2 Ł runs of length L = 2MB
- Step 3 Ł runs of length L = 4MB
- . . . . . .
- Step 10 Ł runs of length L = 1GB (why ?)

Need 10 iterations over the disk data to sort 1GB

Can We Do Better ?

- Hint:

  We have 1MB of main memory, but only used 12KB

Cost Model for Our Analysis

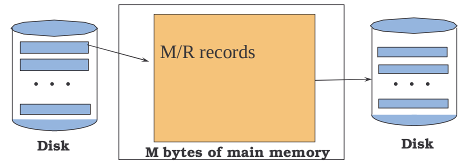
- **B:** Block size (= 4KB)
- **M:** Size of main memory (= 1MB)

For later use (won't need now):

- **N:** Number of records in the file
- **R:** Size of one record

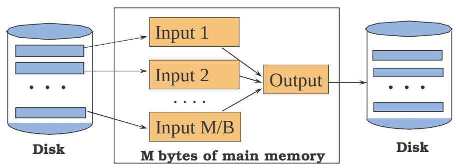
External Merge-Sort

- Phase one: load M bytes in memory, sort
  - Result: runs of length M bytes (1MB)



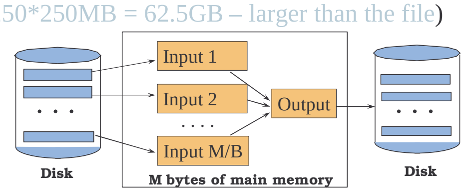
Phase Two

- Merge M/B – 1 runs into a new run (250 runs)
- Result: runs of length M (M/B – 1) bytes (250MB)



Phase Three

- Merge M/B – 1 runs into a new run
- Result: runs of length M $(M/B – 1)^2$ records (250*250MB = 62.5GB – larger than the file)



Need 3 iterations over the disk data to sort 1GB

Cost of External Merge Sort

- Number of passes:

$$1+\lceil \log_{M/B-1} \lceil Size/M \rceil \rceil$$

- How much data can we sort with 10MB RAM?
  - 1 pass Ł 10MB data
  - 2 passes Ł 25GB data (M/B = 2500)
- Can sort everything in 2 or 3 passes !

External Merge Sort

- The **xsort** tool in the XML toolkit sorts using this algorithm
- Can sort 1GB of XML data in about 8 minutes