CS 377
Database Systems
Query Processing

Li Xiong
Department of Mathematics and Computer Science
Emory University
Outline

- Overview
- Algorithms for basic operations
  - Sorting
  - Selection
  - Join
  - Projection
- Query optimization
  - Heuristics
  - Cost-based optimization
Introduction to Query Processing

Figure 19.1
Typical steps when processing a high-level query.

Code can be:
- Executed directly (interpreted mode)
- Stored and executed later whenever needed (compiled mode)
Relational Algebra

- Operators
  - SELECT: $\sigma_{\text{condition}}(R)$
  - PROJECT: $\Pi_{\text{attribute list}}(R)$
  - UNION
  - INTERSECTION
  - SET DIFFERENCE
  - CROSS PRODUCT
  - JOIN

- Expression

- Query tree
Example: SQL query

SELECT title
FROM StarsIn
WHERE starName IN (  
    SELECT name  
    FROM MovieStar  
    WHERE birthdate LIKE ‘%1960’  
);

(Find the movies with stars born in 1960)
Example: Parse Tree

```
SELECT <SelList> FROM <FromList> WHERE <Condition>

<Attribute>    <RelName>            <Tuple> IN <Query>
    title        StarsIn               (  <Query>  )

SELECT <SelList> FROM <FromList> WHERE <Condition>

<Attribute>           <RelName>         <Attribute> LIKE <Pattern>
    name          MovieStar           birthDate            '1960'
```

Example: Initial Logical Query Plan

\[ \Pi_{\text{title}} \]
\[ \sigma_{\text{starName} = \text{name}} \]
\[ \times \]
\[ \Pi_{\text{name}} \]
\[ \sigma_{\text{birthdate} \text{ LIKE} '1960'} \]

MovieStar
Example: Improved Logical Query Plan

\[ \pi_{\text{title}} \]

\[ \sigma_{\text{birthdate LIKE } '1960'} \]

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

\[ \text{StarsIn} \]

\[ \text{MovieStar} \]
Example: A Physical Query Plan

Hash join

SEQ scan
StarsIn

index scan
MovieStar

Parameters: join order, memory size, project attributes,...

Parameters: Select Condition,...
Example: Cost-Based Query Optimization

![Diagram of Cost-Based Query Optimization]

L.Q.P

P1

C1

P2

C2

.....

Pn

.....

Cn

Pick best!
Outline

- Overview
- Algorithms for basic operations
  - Sorting
  - Selection
  - Join
  - Projection
- Query optimization
  - Heuristics
  - Cost-based optimization
Estimate I/O Cost for Implementations

- Count # of disk blocks that must be read (or written) to execute query plan

\[ b(R) = \# \text{ of blocks containing } R \text{ tuples} \]
\[ bfr(R) = \text{max } \# \text{ of tuples of } R \text{ per block} \]
\[ M = \# \text{ memory blocks available} \]
External Sorting

- External sorting: sorting records stored on disk that do not fit entirely in main memory
  - External merge sort
- Internal sorting: sorting data that can fit entirely in main memory
  - Merge sort, quick sort, bubble sort, …
External Merge Sort

Problem

- sort \( r \) records, stored in \( b \) file blocks
- Buffer space: \( M \) blocks (\( M < b \))

Algorithm

- Sorting phase (pass 0): read \( M \) blocks of \( r \) at a time, sort them, write out a level-0 run
  - A run is a sorted subfile
- Merging phase (pass \( i \)): merge \( (M-1) \) level-\((i-1)\) runs at a time, write out a level-\( i \) run
  - Degree of merging = \( M-1 \)
  - \( M-1 \) memory blocks for input, 1 to buffer output
  - Until final pass produces 1 sorted run

Example

- Input: 1, 7, 4, 5, 2, 8, 9, 6, 3, 0 (\( b = 10 \))
- \( M = 3 \)
Performance of External Merge Sort

- Disk I/O - number of block accesses:
  - Each pass reads the entire relation once and write once: $2b$
  - Number of passes?
  - Total I/O?

- Memory requirement: $M$
Performance of External Merge Sort

- Disk I/O - number of block accesses:
  - Each pass reads the entire relation once and write once: $2b$
  - Number of passes: $\log_{M-1}(b/M) + 1$
  - Total I/O: $b \cdot O(b \cdot \log_M b)$

- Memory requirement: $M$
Algorithms for SELECT

- Implementing the SELECT Operation

Examples:

- (OP1): $\sigma_{\text{SSN}=\text{'123456789'}}(\text{EMPLOYEE})$
- (OP2): $\sigma_{\text{DNUMBER}>5}(\text{DEPARTMENT})$
- (OP3): $\sigma_{\text{DNO}=5}(\text{EMPLOYEE})$
- (OP4): $\sigma_{\text{DNO}=5 \text{ AND SALARY}>30000 \text{ AND SEX}=\text{F}}(\text{EMPLOYEE})$
- (OP4'): $\sigma_{\text{DNO}=5 \text{ OR SALARY}>30000 \text{ OR SEX}=\text{F}}(\text{EMPLOYEE})$
- (OP5): $\sigma_{\text{ESSN}=\text{'123456789'} \text{ AND PNO}=10}(\text{WORKS\_ON})$

- Point queries vs. range queries
- Simple selection vs. complex selection
Algorithms for SELECT

- Search Methods for Simple Selection (table scan)
  - **Linear search** (brute force)
    → If the selection condition attribute is not ordered
  - **Binary search:**
    → If the selection condition attribute is ordered
  - **Index search:**
    → Primary index
    → Clustering index
    → Secondary index

- Disk I/O?
- If an index exists, use index search; if the file is ordered, use binary search; otherwise, use linear search
Algorithms for SELECT

- Search Methods for complex Selection:
  - **Conjunctive selection** $\sigma_{DNO=5 \text{ AND } SALARY>30000 \text{ AND } SEX=F}(EMPLOYEE)$
    - Use single index to retrieve the records satisfying some attribute conditions and then check remaining simple conditions
    - What if multiple attributes have an index?
    - Use composite index (or hash structure) directly.
    - Use individual indexes to retrieve record pointers and use intersection of record pointers
  - **Disjunctive selection** $\sigma_{DNO=5 \text{ OR } SALARY>30000 \text{ OR } SEX=F}(EMPLOYEE)$
    - Use linear search if any one condition attribute does not have an access path
    - Use index/binary search when all attributes have access path and then union
Algorithms for JOIN

- Implementing the JOIN Operation:
  - Join (EQUIJOIN, NATURAL JOIN)
    - two-way join: a join on two files
      \[ R \bowtie_{A=B} S \]
    - multi-way joins: joins involving more than two files.
      \[ R \bowtie_{A=B} S \bowtie_{C=D} T \]

- Examples
  - (OP6): EMPLOYEE \bowtie_{DNO=DNUMBER} DEPARTMENT
  - (OP7): DEPARTMENT \bowtie_{MGRSSN=SSN} EMPLOYEE
Join

- Factors affecting performance
  - Tuples of relation stored physically together?
  - Relations sorted by join attribute?
  - Indexes exist?

- Algorithms
  - Nested-loop join (iteration)
  - Sort-merge join
  - Join with index
  - Hash join
Nested Loop Join

- **Iteration join** (conceptually)
  
  for each $r \in R_1$ do
  
  for each $s \in R_2$ do
  
  if $r.C = s.C$ then output $r,s$ pair

- $R_1$: outer table; $R_2$: inner table
Block-based Nested Loop Join (Nested-block join)

- **Iteration join** (conceptually)
  
  for each block $\in R1$ do
  
  for each block $\in R2$ do
  
  output matching pairs

- **R1**: outer table; **R2**: inner table

- What if buffer = $M$ blocks ($M-1$ for reading, 1 for writing)
- Disk I/O?