Query Processing and Optimization

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND
SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

1. Introduction
2. Physical-Query-Plan Operators
3. Query Optimization

Reference:
Chapter 13-14, Database System Concepts, Sixth Edition
Chapter 15, Database System, the Complete Book.

Introduction

THE UNIVERSITY OF AUCKLAND Te Whare Wānanga o Tāmaki Makaurau NEW ZEALAND | SCIENCE DEPARTMENT OF COMPUTER SCIENCE

table course(course_id, title, dept_name);
table instructor(ID, name, dept_name, salary);
table teaches(ID, course_id, sec_id, semester, year);

```
SELECT name, title
FROM instructor
  NATURAL JOIN teaches
  NATURAL JOIN course
WHERE dept_name='Music';
```

- Use the knowledge we have learnt this week to discuss:
  - What are the possible ways to answer the above SQL query?
  - Which one is better?

Introduction

```
SELECT name, title
FROM instructor
  NATURAL JOIN teaches
  NATURAL JOIN course
WHERE dept_name='Music';
```

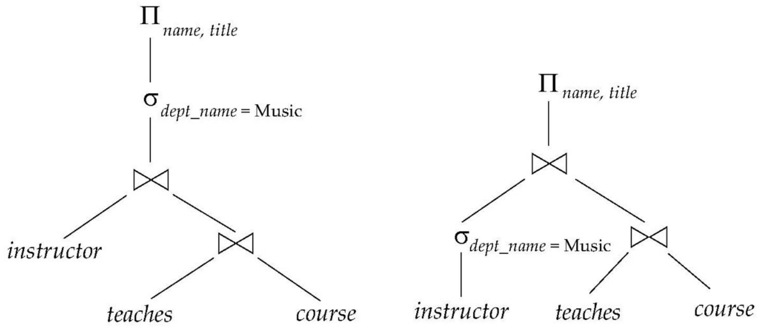
Relational Algebra

- $\Pi_{name,title}(\sigma_{dept\_name=Music}(instructor \bowtie (teaches \bowtie course))$
- $\Pi_{name,title}(\sigma_{dept\_name=Music}(instructor) \bowtie (teaches \bowtie course))$

Introduction

- $\Pi_{name,title}(\sigma_{dept\_name=Music}(instructor \bowtie (teaches \bowtie course))$
- $\Pi_{name,title}(\sigma_{dept\_name=Music}(instructor) \bowtie (teaches \bowtie course))$

Query plan (Logic plan):

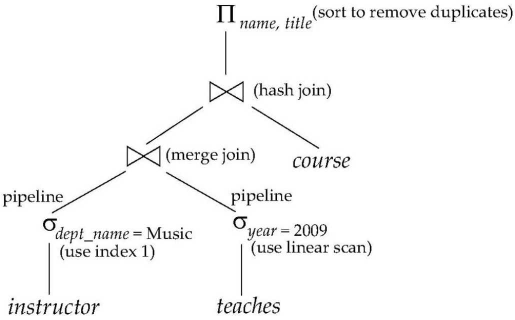


Introduction

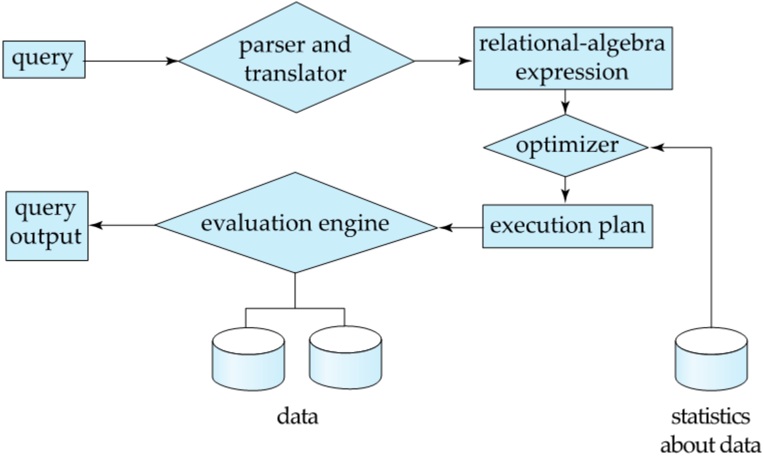
- $\Pi_{name,title}((\sigma_{dept\_name=Music}\, instructor \bowtie \sigma_{year=2009}\, teaches) \bowtie course)$

Physical/Evaluation plan

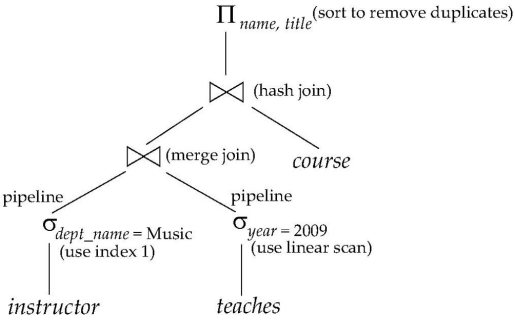
: an annotated expression on detailed evaluation strategy.



Introduction



Introduction



Cost estimation The cost of an operator is dependent on

- Metadata: The size of the underlying relation.
- Implementation: E.g., table scan, or using indexes (B+tree/hashing/$\cdots$).
- reporting: Whether the result needs to be stored on disk.

In this course, we assume that we don't need to store the query results on the disk.

Overview

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND
SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

1. Introduction
2. Physical-Query-Plan Operators
   - One-pass algorithms
   - Multi-pass algorithms
   - Index-based algorithms

Physical-Query-Plan Operators

External Memory Model

Denote by B the block size of the system. Let M be the size of the main memory. The cost of an algorithm is decided by the number of blocks (# of I/Os) transferred between the main memory and the disk.

Sometimes we use $m = M/B$ pages as the size of the main memory to simplify the analysis. In this scenario, the allocation of the memory is in pages.

Physical-Query-Plan Operators

Iterators:

```
Open() {
    b := the first block of R;
    t := the first tuple of block b;
}

GetNext() {
    IF (t is past the last tuple on block b) {
        increment b to the next block;
        IF (there is no next block)
            RETURN NotFound;
        ELSE /* b is a new block */
            t := first tuple on block b;
    } /* now we are ready to return t and increment */
    oldt := t;
    increment t to the next tuple of b;
    RETURN oldt;
}

Close() {
}
```

Physical-Query-Plan Operators

Iterators:

- Combines several operations into one
- Avoids writing temporary files
- Many iterators may be active at one time

Overview

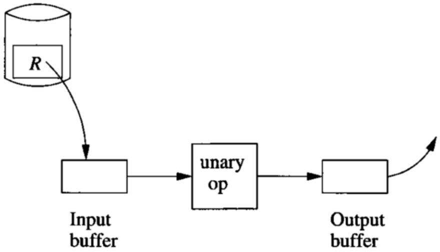
1. Introduction
2. Physical-Query-Plan Operators
   - One-pass algorithms
   - Multi-pass algorithms
   - Index-based algorithms

One-pass algorithms

- Tuple-at-a-time, unary operations: selection and projection
- Full-relation, unary operations: the grouping operator and duplication deduction operator, conditioned that the relation can fit in to the main memory.
- Full-relation, binary operations: unions, intersection, difference, join, product, conditioned that one of the relation fit into the main memory.

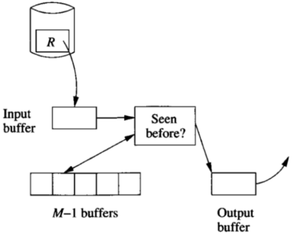
One-pass algorithms

- Tuple-at-a-time, unary operations: selection and projection



One-pass algorithms

- Tuple-at-a-time, unary operations: selection and projection
- Full-relation, unary operations: the grouping operator and duplication deduction operator, conditioned that the relation can fit in to the main memory.



One-pass algorithms

- Tuple-at-a-time, unary operations: selection and projection
- Full-relation, unary operations: the grouping operator and duplication deduction operator, conditioned that the relation can fit in to the main memory.
- Full-relation, binary operations: unions, intersection, difference, join, product, conditioned that one of the relation fit into the main memory.

Arrangement of the memory:

- $M - B - 1$: Hold one relation
- B: buffer one page of the other relation
- 1: buffer for the output

Overview

1. Introduction
2. Physical-Query-Plan Operators
   - One-pass algorithms
   - Multi-pass algorithms
     - Multi-way merge sort
     - Join with both relations larger than the memory size
   - Index-based algorithms

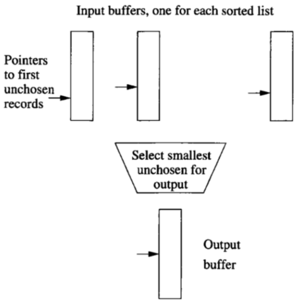
Multi-way Merge Sort

Sort-based algorithms:

- Duplication elimination
- Grouping and aggregation
- Union
- Intersection and difference

Multi-way Merge Sort

Atom operation: merge $m-1$ sorted list into one sorted list by allocating $m-1$ input buffer and 1 output buffer. The cost is linear to the size of the data.



Multi-way Merge Sort

Sort the data of n **blocks on a machine with a memory of** m **blocks.**

- Create runs (pass 0):
  - Each time load m blocks into memory, sort them and then output to create a run
- Merge pass i, $i = 1, 2, \cdots$: for every $m-1$ runs created by pass $i-1$
  - synchronize scan these runs to create a single run
  - one block used for output buffer

  until a pass creates only one run.

Complexity.

$n \log_{m-1} \frac{n}{m} + n \approx n \log_m n$ I/Os

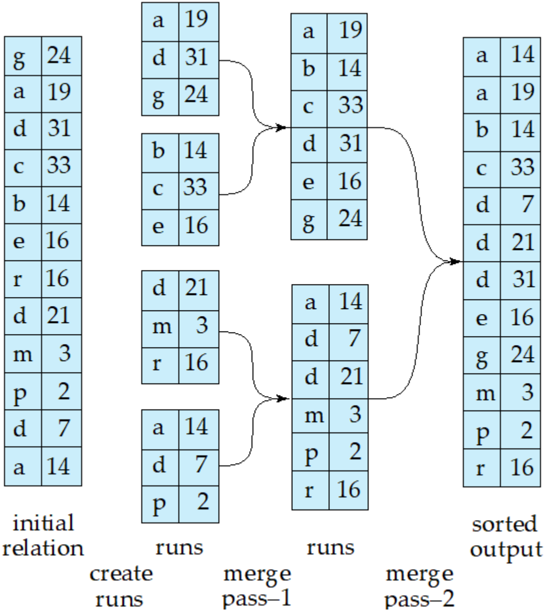


Figure: Example: $m = 3$, blocking factor $= 1$

Join

- Block nested-loop join
- Merge join
- Hash join

Block nested-loop join

$r \bowtie s$: r is stored in n_r blocks. s is stored in n_s blocks

```
for each chunk of m-1 blocks of r
  read these blocks into the main memory buffers;
  organize their tuples into a search structure whose search key is the
   ↪  common attributes of r and s;
  for each block bs of s
    load bs into the main memory
      for each tuple t of bs
        find the tuples of r in the buffer that join with t
        report the result
```

Complexity: $\lceil n_r/(m-1) \rceil n_s + n_r$ I/Os.

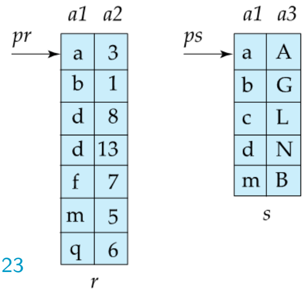
Merge Join

$r \bowtie s$: r is stored in n_r blocks. s is stored in n_s blocks.

Join attribute: A.

Assume that r and s are both ordered A.

```
for each value x of attribute A shared by s and r (synchronize scan)
  block-nested-loop join on subrelations s[A=x] and r[A=x]
```



Complexity.

- Best case: $n_r + n_s$
- Worst case: block-nested-loop-join
- Consider the cost when
  - A is a key of r, or
  - all the attribute values are having the same frequency

Hashing: Partition a relation r to k buckets.

Consider a relation r with an attribute A and an integer $k \in [1, m)$;
Hash function $h_A()$ maps a tuple t to an integer in range $[0, k)$ based on its value on A.

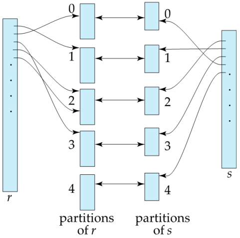
```
initialize k buckets using k empty buffers;
for each block b of relation r
  read block b into the (k+1)-th buffer;
  for each tuple t in b
  if the buffer for bucket h(t) has no room for t
    write the buffer to disk;
    initialize a new empty block in that buffer;
    copy t to the buffer for bucket h(t);
for each bucket do
  if buffer for this bucket is not empty
    write the buffer to disk;
```

Denote by r_i the i-th bucket of r.

Hash Join

$r \bowtie s$: r is stored in n_r blocks. s is stored in n_s blocks.
Join attribute: A.
Hash function $h_A()$ maps from a tuple to an integer range $[0, k)$ based on A.

1. Partition r to k buckets using hash function h_A
2. Partition s to k buckets using hash function h_A
3. For each integer $i \in [0, k)$
   - Join on r_i and s_i (one pass/multi pass)



Examples: Cost Estimation

Goal is to count disk I/Os. But we First have to estimate sizes of intermediate results?

- V(A,r) number of distinct values of attribute A in relation r.
- |r| number of tuples in relation r.

Consider relation $r(A, B)$ with $n_r = 1000$ blocks and relation $s(A, C)$ with $n_s = 500$ blocks. The memory had $m = 101$ pages.

- Nested-loop-join
- Multi-way merge join
- Hash join

Examples: Cost Estimation, Nested-Loop-Join

Consider relation $r(A, B)$ with $n_r = 1000$ blocks and relation $s(A, C)$ with $n_s = 500$ blocks. The memory had $m = 101$ pages.

- Outer-loop: load chunk of 100 blocks of s to the main memory
  5 chunks, 100 blocks each
- Inner-loop: scan r in 1000 blocks
- Total cost 5500 I/Os.

Switch the inner and outer loop:

- Outer-loop: load chunk of 100 blocks of r to the main memory 10 chunks, 100 blocks each
- Inner-loop: scan r in 500 blocks
- Total cost 6000 I/Os.

Conclusion: slight advantage in having the smaller relation on the outer loop.

Examples: Cost Estimation, Merge Join

Consider relation $r(A, B)$ with $n_r = 1000$ blocks and relation $s(A, C)$ with $n_s = 500$ blocks. The memory had $m = 101$ pages.

- The sorting cost of r: 4000 (two reads and two writes per block)
- The sorting cost of s: 2000 (two reads and two writes per block)
- Merge join: If A is the key of one relation, then the cost is 1500 blocks.
- Total cost 6500 I/Os.

Why linear time (when A is a key) merge join is not as good as quadratic time nested loop join in this case?

Examples: Cost Estimation, Hash Join

Consider relation $r(A, B)$ with $n_r = 1000$ blocks and relation $s(A, C)$ with $n_s = 500$ blocks. The memory had $m = 101$ pages. Let $k = 100$.

- The average size for each bucket is 10 blocks for relation r and 5 for s.
- Partition n and s (linear time): $1500 \times 2 = 3000$ I/Os
- Since for each $i \in [0, k)$, the buckets of r and s can altogether fit in main memory, one-pass join: 1500 I/Os for loading.
- Total cost: 4500 I/Os.

Overview

1. Introduction
2. Physical-Query-Plan Operators
   - One-pass algorithms
   - Multi-pass algorithms
   - Index-based algorithms

Index-based Algorithms

Application of Index in

- Selection
- Join

To avoid/reduce the number of table scan.

Index-based Algorithms: Cost Estimation Assumptions

We first have to estimate sizes of the intermediate results.

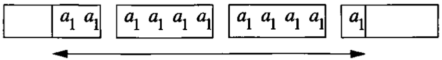
- $V(A, r)$ number of distinct values of attribute A in relation r.
- $|r|$ number of tuples in relation r.
- n_r number of blocks used to store relation r.

Important assumptions:

- Simple selection: All $V(A, r)$ values are equally likely for attribute A; in other words, $|\sigma_{A=c} r| = |r|/V(A, r)$.
- Selection involving inequality $|\sigma_{A<c} r|$: Common assumption that $1/3$ will meet the condition.
- Complex conditions AND: use decompositions.
  - $|\sigma_{A=c \text{ and } B<d} r| = |r|/3V(A, r)$.

Index-based Algorithms: Selection

Sequential file:



The actual cost of $\sigma_{A=v}r$ can be slightly larger than $n_r/V(A,r)$.

- The index is not kept entirely in main memory, some disk I/Os are need to support the index lookup.
- Even when all the tuples with $A = v$ might fit in b blocks, they could be spread over $b+1$ blocks because they don't start at the beginning of a block.
- The blocks can be not full, e.g., the B^+tree's leaf nodes can be not full.

Unordered file

- We assume that we need to visit $|r|/V(A,r)$ blocks for answering $\sigma_{A=v}r$.

Index-based Algorithms: Selection

Assume that for relation $r(A, B)$, $n_r = 1000$ and $|r| = 20,000$. Consider $\sigma_{A=0} r$. We ignore the cost accessing the index blocks in all cases.

- r is sequential on A but we do not use the index — 1000 I/Os.
- r is unordered on A but we use an index on A — 20,000 I/Os.
- r is sequential on A, $V(A, r) = 100$, use an index — 10 I/Os.
- r is unordered on A, $V(A, r) = 10$, use an index — 2000 I/Os.
- A is the key of r, use an index — 1 I/O.

Index-based Algorithms: Nested-Loop Join

$r(A, B) \bowtie s(A, C)$:

- r has $|r|$ tuples on n_r blocks, s is stored in n_s blocks
- s has an index on A;

```
for each tuple t of r
  Get all tuples of s that can join with t using the index
```

Complexity: For each tuple t of r, an average of $|s|/V(A, s)$ tuples should be retrieved, the cost is dependent on s

- If s is sequential, for each t the cost is $n_s/V(A, s)$ I/Os;
- Otherwise, the cost is $|s|/V(A, s)$ I/Os.

Index-based Algorithms: Nested-Loop Join

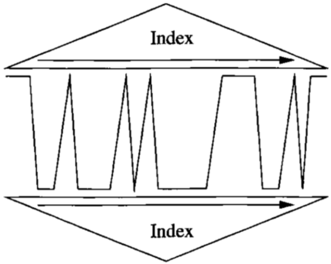
Example: Consider $r(A, B)$ with $n_r = 1000$ and $s(A, C)$ with $n_s = 500$ while ten tuples of either relation fit in one block. Therefore, $|r| = 10,000$ and $|s| = 5000$. Assume that $V(A, s) = 100$. Suppose that s is ordered on A and there is a clustering index of s on A. Compute the cost of $r \bowtie s$.

- The number of I/Os in accessing s is $10000 \times 500/100 = 50000$ I/Os.

If r is considerably smaller than s then the index-based nested-loop join would be better than nested-loop join.

Index-based Algorithms: Merge Join

Both r and s are sequentially stored, each having a clustered index on A.



Overview

1. Introduction
2. Physical-Query-Plan Operators
   - One-pass algorithms
   - Multi-pass algorithms
   - Index-based algorithms
3. Query Optimization

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

SCIENCE

Thank you for your attention!

Any questions?