CS 133: Databases

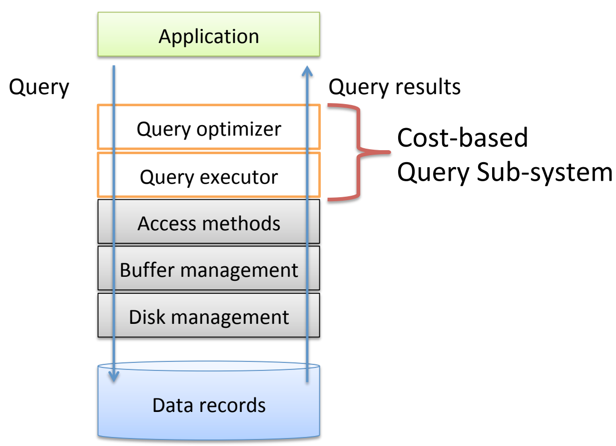
Fall 2019
Lec 10 – 10/08
Query Evaluation

Prof. Beth Trushkowsky

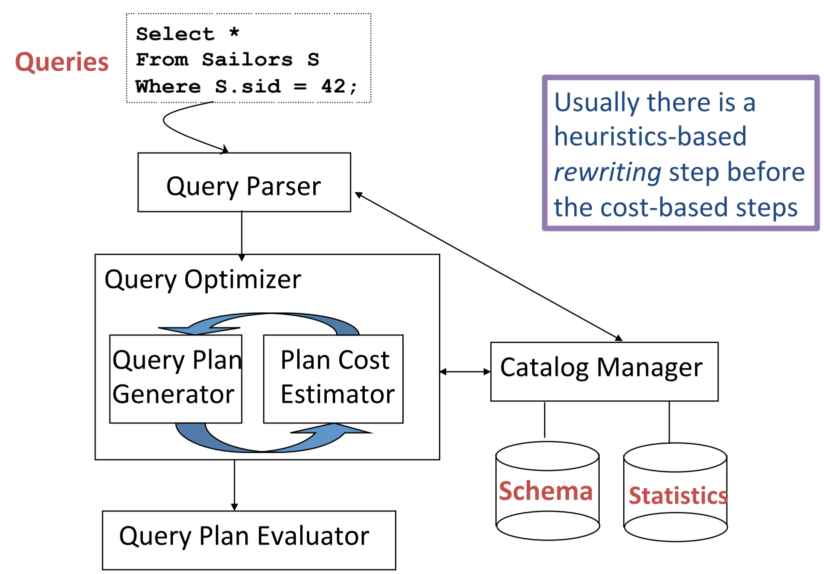
Goals for Today

- Learn about the components of query processing and idea of **different algorithms** for relational operators
- Understand the importance of **out-of-core** a.k.a. **external** sorting and hashing algorithms
- Reason about the I/O cost of sorting or hashing algorithms given size of relations and available buffer pool space

Simplified RDBMS Architecture



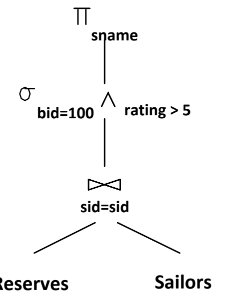
Cost-based Query Sub-System



Logical Query Plan: Example

- Example SQL query:

```
SELECT S.sname
FROM   Reserves R, Sailors S
WHERE  R.sid=S.sid
   AND R.bid=100 AND S.rating>5
```



Logical query plan tree with relational operators

- Equivalent Relational Algebra expression:

$\pi_{sname}\ \sigma_{bid=100 \wedge rating > 5}(\text{Reserves} \bowtie_{sid=sid} \text{Sailors})$

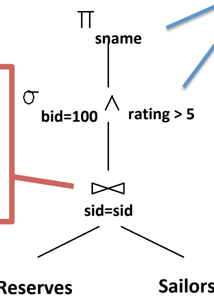
Logical Plan to Physical Plan

- Logical query plan *partially* shows us how to evaluate query
  - Missing: choice of **specific algorithm** for *executing* operators
  - Also: ordering of operators
- How to choose?!
  - No one algorithm always best
  - *Cost-based optimization*
  - Forms *physical* query plan

Selections: how to apply multiple constraints? *Projections*: duplicate elim.?

Join algorithm choice:
- Simple nested loop
- Index nested loop
- **Sort-merge?**
- …

Access path choice: Sequential scan? Indexes?



Implementing the `Project` Operator

- Suppose we do not care about removing *duplicates*

```
SELECT R.attribute FROM R;
```

- How many I/Os? What would this process look like (with respect to the disk and buffer pool)?

Read in pages of *R* one at a time, remove unwanted fields in ***one pass*** over *R*

External Algorithms: Sorting and Hashing

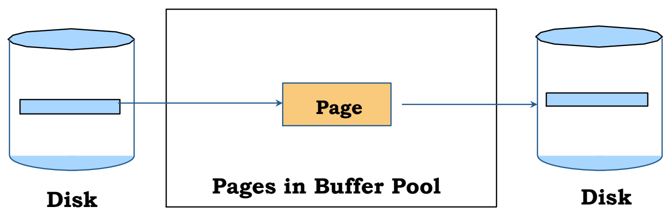
- In various parts of a query plan, important to get "same" tuples *together*
  - DISTINCT (duplicate elimination)
  - GROUP BY (form the groups)
  - Sort-merge JOIN algorithm
  - ORDER BY (user wants output sorted)

i.e., co-resident in memory (buffer pool)

- ***Problem***: sort 100GB of data with 1GB of RAM
- Solution: out-of-core (external) algorithms that ***divide and conquer***
  - Idea: intelligent use of available buffer pool space

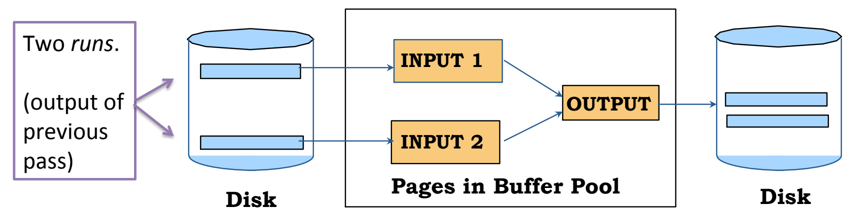
Two-Way Sort

- Algorithm operates in a sequence of *passes*
- Pass 0 -- For all pages in file:
  - Read page, sort it in RAM, write the sorted page to disk (don't overwrite original).
  - Only one buffer page is used in this pass
  - Each sorted page output called a sorted ***run***



Two-Way Sort: Passes 1, 2, …

- Pass 1, 2, …, etc. (merging):
  - Requires **three buffer pages**: two input, one output
  - **Merge** pairs of runs into runs *twice as long*



Two-Way External Merge Sort

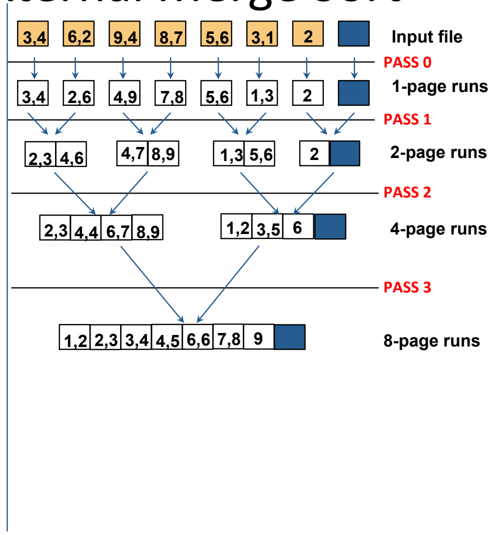
- Each pass: read + write each page in file.
- *N* pages in the file => number of passes?

$$= \lceil \log_2 N \rceil + 1$$

- So total cost is?

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

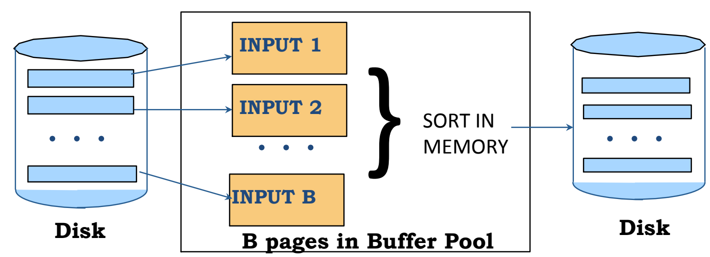
- <u>*Idea:*</u> ***Divide, conquer, merge***



General External Merge Sort

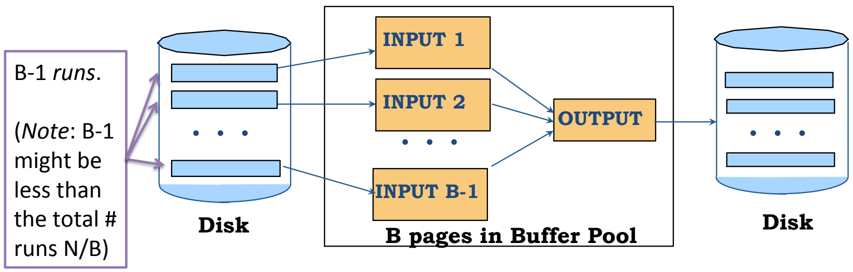
We have more than 3 buffer frames. How can we utilize them?

- To sort a file with N pages using B buffer frames
  - Pass 0: use *B* buffer pages. Produce $\lceil N / B \rceil$ sorted runs of *B* pages **each**



General External Merge Sort: Passes 1, 2, …

- In each of Pass 1, 2, etc.: ***merge*** B-1 *runs*
  - Creates runs of (B-1) * (size of runs from previous pass)



Cost of External Merge Sort

- Cost = 2N * (# of passes)
  - In each pass, read and write each page of file
  - (N is size of relation in pages)
- Try Exercise (2-3)
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: $\lceil 108 / 5 \rceil$ = 22 sorted runs of 5 pages each (last only 3)
  - Pass 1: $\lceil 22 / 4 \rceil$ = 6 sorted runs of 20 pages each (last only 8)
  - Pass 2: yields 2 sorted runs, 80 pages and 28 pages
  - Pass 3: yields one sorted run of 108 pages
- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
  - Formula check: $1 + \lceil \log_4 22 \rceil = 1 + 3 \rightarrow$ 4 passes

Number of Passes with External Sort (with *B* Buffer Frames and *N* pages)

N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Algorithm for *Internal* Sort

- Quicksort is a fast way to sort in memory.
- An alternative is "tournament sort" (a.k.a. "heap sort")
  - Idea: create **initial sorted runs** that **are *longer* than B** pages
  - See book for more…

Average length of a run is ~*2B* ← **Why would we care?**

Sort: Kind of a Big Deal

	Daytona
Gray	2016, 44.8 TB/min **Tencent Sort** 100 TB in 134 Seconds 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub
Cloud	2016, $1.44 / TB **NADSort** 100 TB for $144 394 Alibaba Cloud ECS ecs.n1.large nodes x (Haswell E5-2680 v3, 8 GB memory, 40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk) Qian Wang, Rong Gu, Yihua Huang Nanjing University Reynold Xin Databricks Inc. Wei Wu, Jun Song, Junluan Xia Alibaba Group Inc.

(source: sortbenchmark.org)

Reasoning about Passes

- Exercise 4:
  - How big of a relation can we sort in two passes?
  - B(B-1) pages

Projection (with Duplicate Elim.)

```
SELECT DISTINCT R.sid, R.bid
FROM    Reserves R
```

- Suppose Reserves is 1000 pages, *sid* and *bid* together are 25% of each record
- Basic approach with sorting:
  1. Scan R, extract only the needed fields
  2. Sort the resulting set
  3. Read in, removing duplicates which will be adjacent
- Cost: Reserves with size ratio 0.25 = 250 pages
  Using 20 buffer pages can sort in 2 passes, (ignores cost of final output):

  (1000 + 250) + (2 * 2 * 250) + 250 = 2500 I/Os

- Can improve using modification of external sort algorithm…

Exercise 5

- Can improve Duplication Elimination by modifying external sort algorithm:
  1. Modify Pass 0 of external sort to eliminate unwanted fields.
  2. Modify merging passes to eliminate duplicates.
- Cost of improved version:
  - Pass 0: read 1000 pages, write out 250 in 13 runs of 20 pages
  - Pass 1: merge runs by reading in 250
  - Total I/Os: **1000+250+250 = 1500s**

Alternative: Hashing

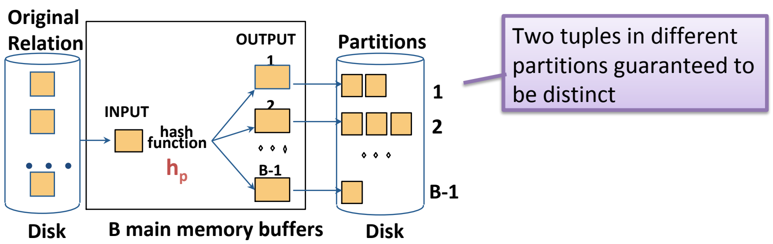
- We do not always require *order* for tuples
  - Removing duplicates
  - Forming groups
- Just need "like" things to be together
  - Hashing!
  - But how to build **hash table without staying in RAM**?

External Hashing: Divide and Conquer

- ***Divide***: Use a hash function h_p to separate records into **disk-based** partitions
- ***Conquer***: Read partitions into RAM-based hash table one at a time
  - For each partition, hash with *another* hash function h_r
- Note: Two different hash functions: h_p is coarser-grained than h_r

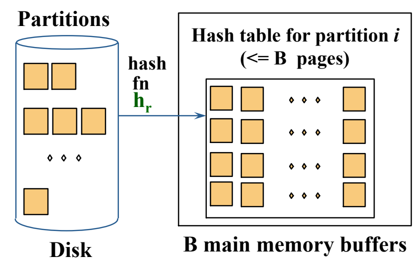
Projection: DupElim Based on Hashing

- Partition phase:
  - Read relation using one input buffer frame, retaining only necessary fields for projection
  - Hashing on h_p to yield B-1 partitions



Projection: DupElim Based on Hashing

- Duplicate Elimination phase
- For each partition:
  - Read in pages
  - Build an in-memory hash table, using second hash function h_r, and **eliminating duplicates as you go**
- If a partition does not entirely fit in buffer pool, need to *recursively* partition before this phase



Note: ignoring small overhead in memory for hash data structure

Example: Hashing DupElim

- Cost for Projection with DupElim using hashing?
  - Assume each of the partitions formed in first pass fits in buffer pool…
- For *Reserves* query:
  - Read **1000** pages
  - Write out partitions of projected tuples
    - **250** pages, because 25% of record retained
  - Read and do duplicate elimination on each partition
    - total **250** page reads
- Total : **1000** + **250** + **250** = 1500 I/Os.