RDBMS and SQL

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Lecture 10, 4 November 2026



Query processing

- Translate the query from SQL into relational algebra
- Evaluate the relational algebra expression

```
select salary
from instructor
where salary < 75000
```

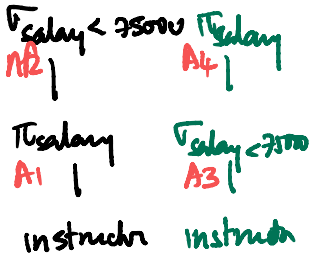
Query processing

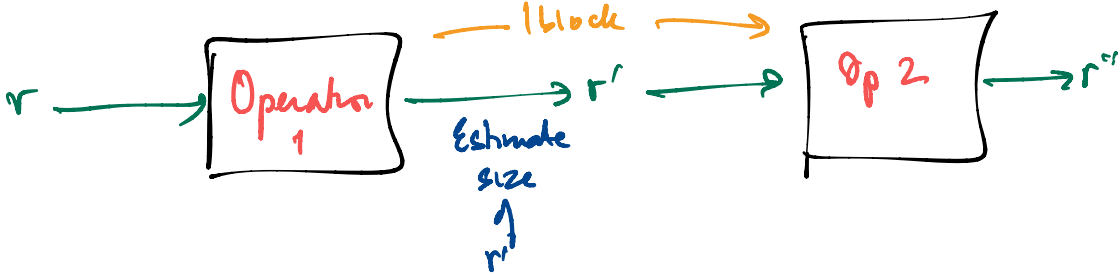
- Translate the query from SQL into relational algebra
- Evaluate the relational algebra expression
- Challenges
  - Many equivalent relational algebra expressions

    $\sigma_{salary<75000}(\pi_{salary}(instructor))$ vs $\pi_{salary}(\sigma_{salary<75000}(instructor))$
  - Many ways to evaluate a given expression

Query processing

- Translate the query from SQL into relational algebra
- Evaluate the relational algebra expression
- Challenges
  - Many equivalent relational algebra expressions
    $\sigma_{salary<75000}(\pi_{salary}(instructor))$ vs $\pi_{salary}(\sigma_{salary<75000}(instructor))$
  - Many ways to evaluate a given expression
- Query plan
  - Annotate the expression with a detailed evaluation strategy key values
    - Use index on *salary* to find instructors with $salary < 75000$
    - Or, scan entire relation, discard rows with $salary \geq 75000$





Pipeline

Query optimization

- Choose plan with lowest cost
- Maintain database catalogue — number of tuples in each relationn, size of tuples, ...
- Assess cost in terms of disk access and transfer, CPU time, ...
- For simplicity, ignore in-memory costs (CPU time), restrict to disk access

ignore

seek $\approx$ 10 x transfer

Query optimization

- Choose plan with lowest cost
- Maintain database catalogue — number of tuples in each relationn, size of tuples, ...
- Assess cost in terms of disk access and transfer, CPU time, ...
- For simplicity, ignore in-memory costs (CPU time), restrict to disk access
- Disk accesses
  - Relation r occupies b_r blocks
  - Disk seeks — time t_S per seek
  - Block transfers — time t_T per transfer

$t_S \approx 10 \cdot t_T$

Query optimization

- Choose plan with lowest cost
- Maintain database catalogue — number of tuples in each relationn, size of tuples, ...
- Assess cost in terms of disk access and transfer, CPU time, ...
- For simplicity, ignore in-memory costs (CPU time), restrict to disk access
- Disk accesses
  - Relation r occupies b_r blocks
  - Disk seeks — time t_S per seek
  - Block transfers — time t_T per transfer
- Other factors — buffer management etc



Selection

- Linear search

$\sigma_\theta(r)$

Read $1 \cdot t_S + b_r \cdot t_T$

Ignore final write

Selection

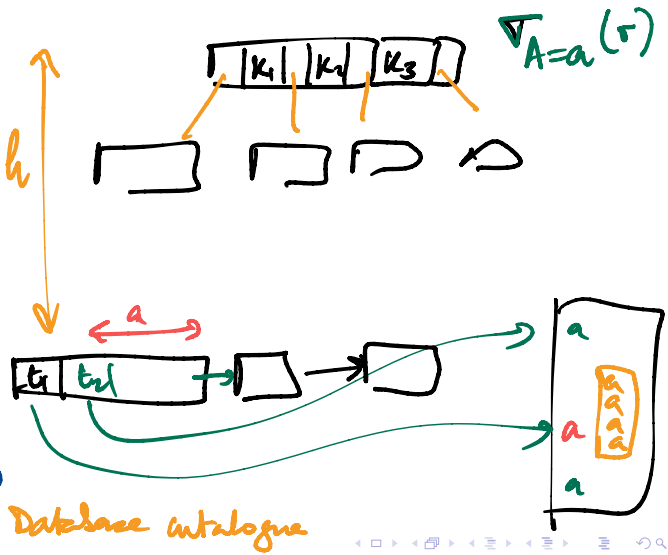
- Linear search
- B$^+$-tree index — index height h_i
  - Equality on key
  - Equality on nonkey

$h+1$ reads
$h+1$ seeks

$(h+1)(t_S + t_T)$

$h(t_S + t_T)$ index lookup
$(t_S + t_T)$ · # of a

Database catalogue



Selection

- Linear search
- B^+-tree index — index height h_i
  - Equality on key
  - Equality on nonkey
- Secondary B^+-tree index (key, non-key)

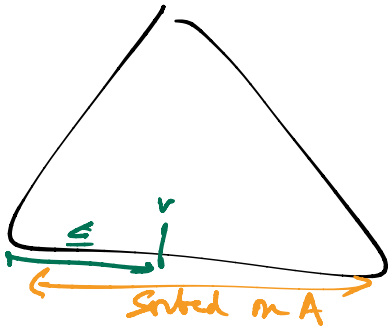


Table is not ordered with this value

Selection

- Linear search
- B$^+$-tree index — index height h_i
  - Equality on key
  - Equality on nonkey
- Secondary B$^+$-tree index (key, non-key)
- B$^+$-tree index, comparison on A
  - Sorted on A
  - Not sorted on A





$$\frac{x}{y} \cdot n_r$$

Selection

- Linear search
- B^+-tree index — index height h_i
  - Equality on key
  - Equality on nonkey
- Secondary B^+-tree index (key, non-key)
- B^+-tree index, comparison on A
  - Sorted on A
  - Not sorted on A
- Boolean combinations
  - Conjunctive selection using one index
  - Conjunctive selection using composite index
  - Disjunction, negation, ...

$\sigma_{\theta_1 \wedge \theta_2}(r)$

Filter on θ_1
& check θ_2

Sorting

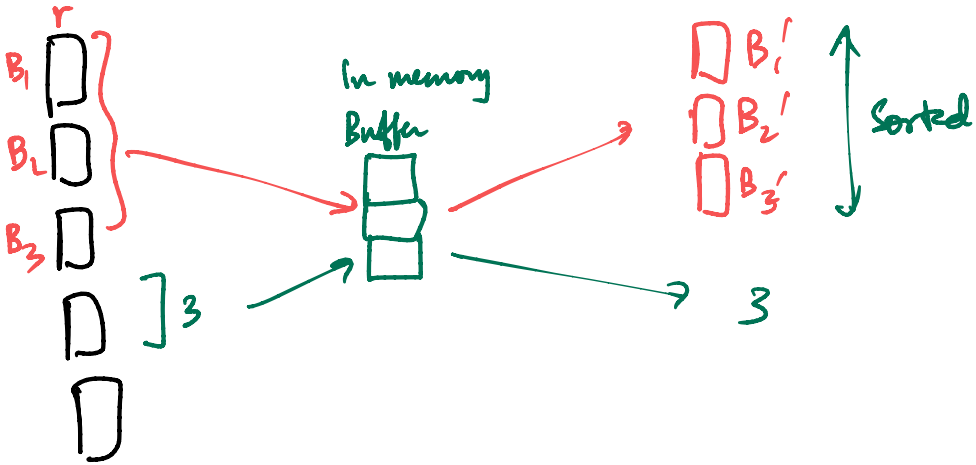
- In-memory sorting vs sorting on disk

Sorting

- In-memory sorting vs sorting on disk
- Traditional merge sort

External merge sort

- N records, b_r blocks, M blocks in memory



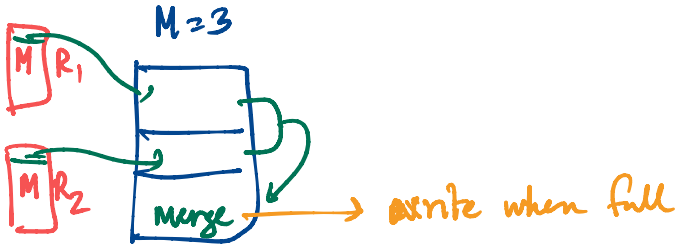
External merge sort

- N records, b_r blocks, M blocks in memory

$b_r \gg M$

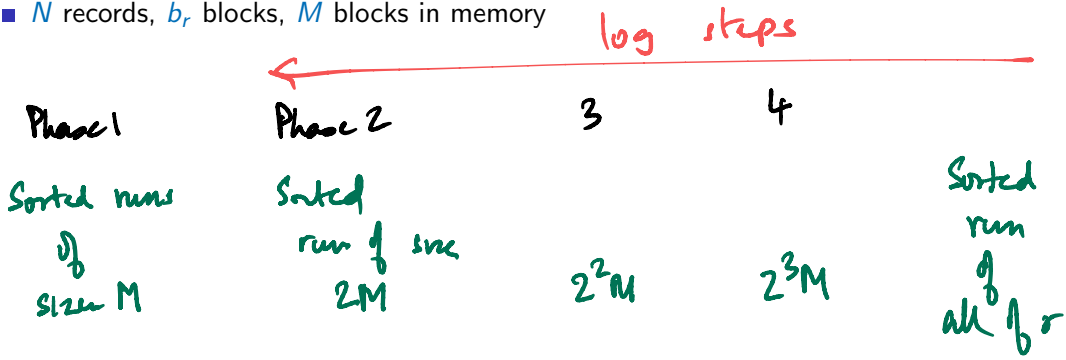
Phase 1

Create sorted "runs" of size M



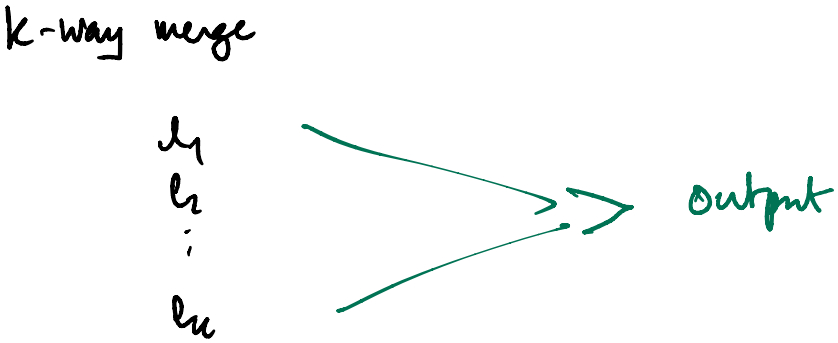
External merge sort

- N records, b_r blocks, M blocks in memory



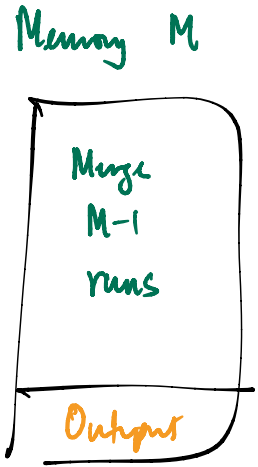
External merge sort

- N records, b_r blocks, M blocks in memory



External merge sort

- N records, b_r blocks, M blocks in memory



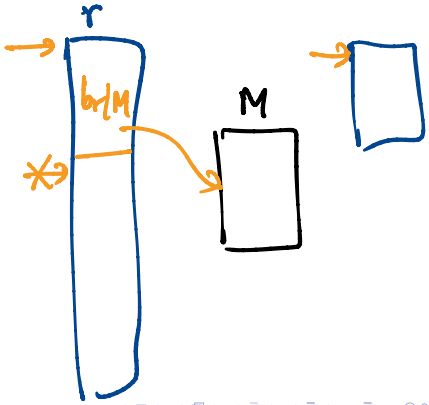
External merge sort

- N records, b_r blocks, M blocks in memory

Ph 1 Read $\frac{b_r}{M}$ blocks into memory, sort & write

b_r reads & b_r writes

$\frac{b_r}{M}$ seeks & $\frac{b_r}{M}$ seeks



External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M

External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs

External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs
- Complexity
  - b_r/M sorted runs, $\lceil \log_{M-1}(b_r/M) \rceil$ merge passes

External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs
- Complexity
  - b_r/M sorted runs, $\lceil \log_{M-1}(b_r/M) \rceil$ merge passes
  - Block transfers — $b_r\ (2\lceil \log_{M-1}(b_r/M) \rceil + 1)$
    - Why not $b_r\ (2\lceil \log_{M-1}(b_r/M) \rceil + 2)$?

Ignore final write

P1 $b_r + b_r$ transfers

P2 - Plog$_{M-1}$ - $b_r + b_r$ trans

External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs
- Complexity
  - b_r/M sorted runs, $\lceil \log_{M-1}(b_r/M) \rceil$ merge passes
  - Block transfers — $b_r \ (2\lceil \log_{M-1}(b_r/M) \rceil + 1)$
    - Why not $b_r \ (2\lceil \log_{M-1}(b_r/M) \rceil + 2)$?
  - Block seeks — $2\lceil b_r/M \rceil + b_r \ (2(\lceil \log_{M-1}(b_r/M) \rceil - 1)$

Computing joins

- Running example
  - $Student \bowtie Takes$

Computing joins

- Running example
  - *Student* $\bowtie$ *Takes*
  - *Student* — 5000 rows, 100 blocks
  - *Takes* — 10000 rows, 400 blocks

50 rows / block
25 rows / block

Nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)

r_1 r_2

100 reads for each t_1 in Student
  for each t_2 in Takes
    {

Once per row in Student
  400 blocks read
  1 seek

b_{r_1} reads + $n_{r_1} \cdot b_{f_2}$ reads

b_{r_1} seeks + n_{r_1} seeks

Nested-loop join

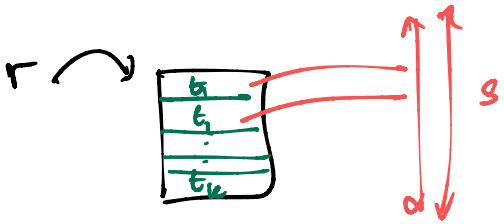
- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation

Nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + n_r \cdot b_s$

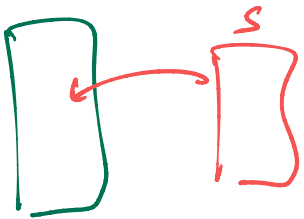
Nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + n_r \cdot b_s$
  - Block seeks: $b_r + n_r$ — inner relation read sequentially



Nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + n_r \cdot b_s$
  - Block seeks: $b_r + n_r$ — inner relation read sequentially
  - Special case: smaller relation fits in memory



Block nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)

Block nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation

Block nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + b_r \cdot b_s$

$= \text{vs. } n_r \cdot b_s$

Block nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + b_r \cdot b_s$
  - Block seeks: $b_r + b_r = 2b_r$



Indexed nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)

Indexed nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation

Indexed nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Total cost: $b_r(t_T + t_S) + n_r \cdot c$
    - c is cost of single selection on s

same

If not key, multiply by estimated multiplicity of value

$R_1 \cap R_2 \rightarrow R_1$

or

$R_1 \cap R_2 \rightarrow R_2$

Merge join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)

Merge join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
    - Assume relations are sorted (add cost of sorting)

Merge join

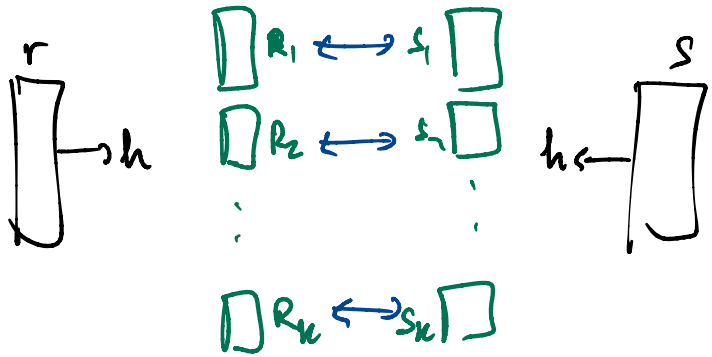
- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
    - Assume relations are sorted (add cost of sorting)
  - Block transfers: $b_r + b_s$

Merge join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
    - Assume relations are sorted (add cost of sorting)
  - Block transfers: $b_r + b_s$
  - Block seeks: $\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil$
    - Read a chunk of blocks b_b at a time

Hash join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Hash function on join attribute A, n_h output values



Hash join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Hash function on join attribute A, n_h output values
- Join each pair of hash buckets — build index and probe

