RDBMS and SQL

Madhavan Mukund

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

Lecture 11, 5 November 2024

Query processing

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

Query processing

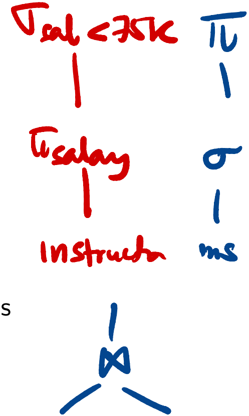
- Translate the query from SQL into relational algebra
- Evaluate the relational algebra expression
- Challenges
  - Many equivalent relational algebra expressions

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

Query 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 $salary \geq 75000$



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

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

$$t_S + \underline{b_r * t_T}$$

Sequentially organized

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

(A1) Linear search — $t_S + b_r * t_T$

(A2) Clustering index, equality on key — index height $h_i * (t_S + t_T)$ $\sigma_\theta(r)$

$+1$

(A3) Clustering index, equality on nonkey

(A4) Secondary index (key, non-key)

$\swarrow K_1 \downarrow K_2 \downarrow \cdots \downarrow K_{n-1} \searrow$

(A5) Clustering index, comparison — sorted on A

(A6) Clustering index, comparison — not sorted on A

(A7) Conjunctive selection using one index

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

(A8) Conjunctive selection using composite index

(A9) Conjunctive selection using intersection of pointers

$\sigma_{\theta_1}(r)$ — check θ_2

(A10) Disjunctive selection by union of pointers

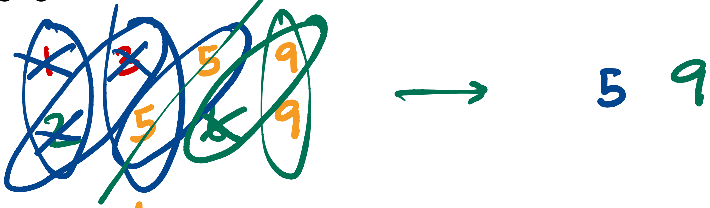
(Neg) Negation

Sorting

- In-memory sorting vs sorting on disk

Sorting

- In-memory sorting vs sorting on disk
- Merging sorted lists — varieties



Common elements

Union

Intersection

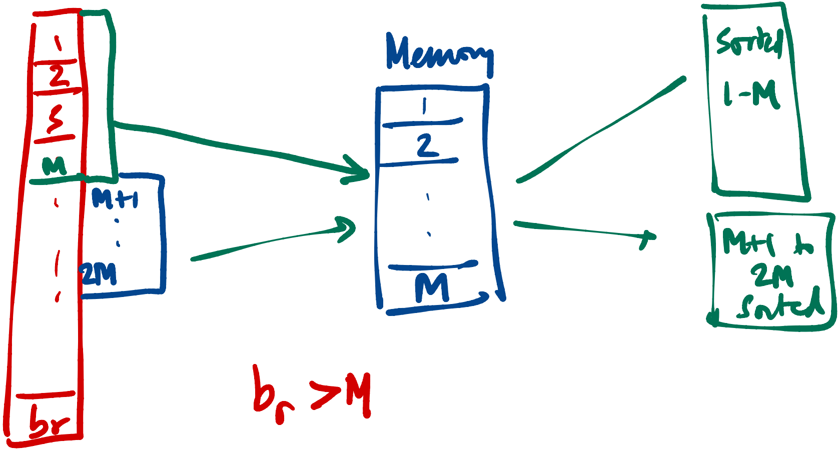
List
difference

Sorting

- In-memory sorting vs sorting on disk
- Merging sorted lists — varieties
- Traditional merge sort

External merge sort

- N records, b_r blocks, M blocks in memory



$b_r > M$

First pass

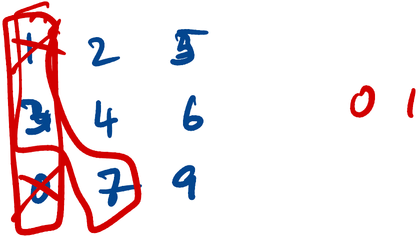
Created sorted "runs" of size M

$\left\lceil \frac{b_r}{M} \right\rceil$

Such runs

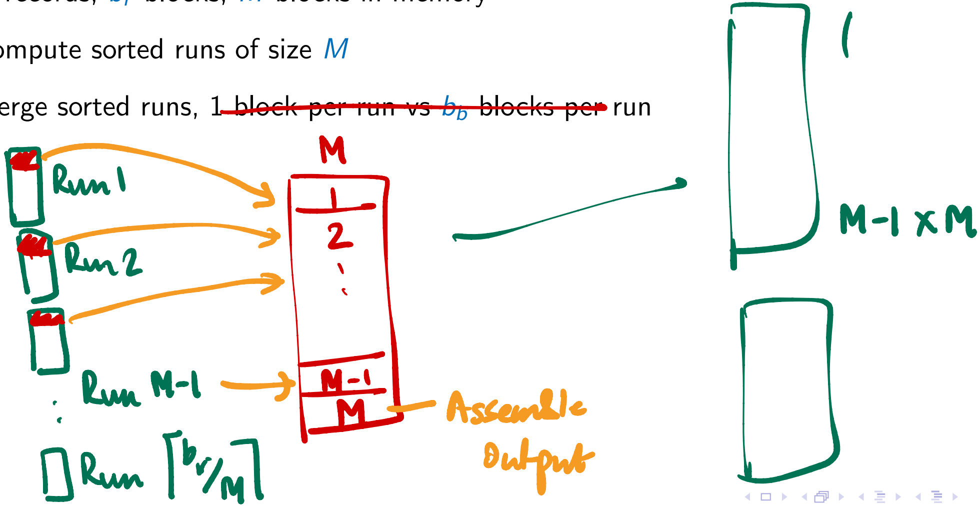
External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M — $\left\lceil \frac{b_r}{M} \right\rceil$



External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs, 1 ~~block per run vs b_b blocks per~~ run



External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs, 1 block per run vs b_b blocks per run

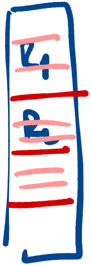
$$\begin{tikzcd}
b_r \arrow[d] \\
\left\lceil b_r/M \right\rceil \arrow[d] \\
b_r/M^2 \arrow[d, phantom, "\vdots"] \\
b_r/M^k < M
\end{tikzcd}$$

External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs, 1 block per run vs b_b blocks per run
- Complexity
  - b_r/M sorted runs, $\lceil \log_{\underline{\underline{\lfloor M/b_b \rfloor - 1}}}(b_r/M) \rceil$ merge passes

$\uparrow$

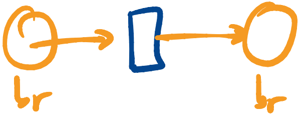
M-1 if $b_b = 1$

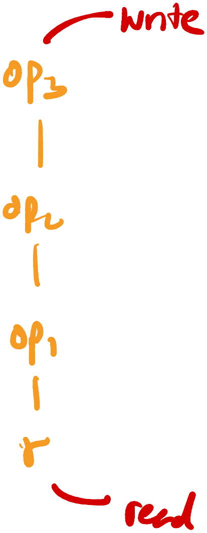


External merge sort

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

Final output is not counted





External merge sort

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

Computing joins

- Running example
  - $Student \bowtie Takes$

Computing joins

- Running example
  - *Student* ⋈ *Takes*
  - *Student* — 5000 rows, 100 blocks   50 rows / block
  - *Takes* — 10000 rows, 400 blocks   25 rows / block

```
for x in Student
    for y in Takes
        ≡
```

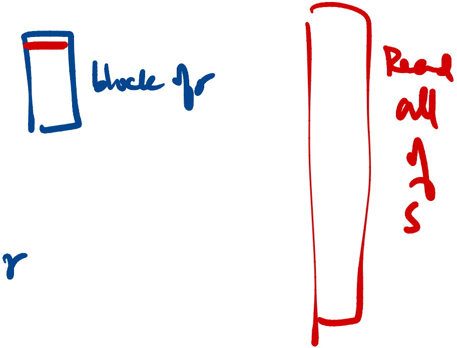
Nested-loop join

- (5000 rows, 100 blocks) $\mathit{Student} \bowtie \mathit{Takes}$ (10000 rows, 400 blocks)

Nested-loop join

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

```
for x in r
    for y in s
```



n_r

rows in r

$b_s \times n_r + b_r$

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: $b_r + n_r \cdot b_s$

r = Student

s = Takes

$100 + 5000 \times 400 = 20 \times 10^5 + 100 = 2000100$

$400 + 10000 \times 100 = 10 \times^{6} + 400 = 1000400$

Nested-loop join

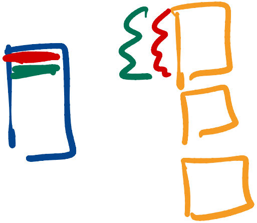
- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + n_r \cdot b_s$
  - Block seeks: $b_r + n_r$ — inner relation read sequentially

Seeking s once per iteration

Each block requires a fresh seek

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 – make inner

Block nested-loop join

- (5000 rows, 100 blocks) $\mathit{Student} \bowtie \mathit{Takes}$ (10000 rows, 400 blocks)

Block nested-loop join

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

n_r

Indexed nested-loop join

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

Index on Takes exists (on ID)

for each row in Students

look up index on Takes

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* ⋈ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Total cost: $b_r(t_T + t_S) + n_r \cdot c$

    index lookup
    - c is cost of single selection on s

Merge join

- (5000 rows, 100 blocks) *Student* ⋈ *Takes* (10000 rows, 400 blocks)

Sort r

Sort s

Merge (intersect)

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

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
    - 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
  - Block transfers: $b_r + b_s$
  - Block seeks: $\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil$

+ Sorting cost

↑

Chunks of b_b are read at a time

Merge join

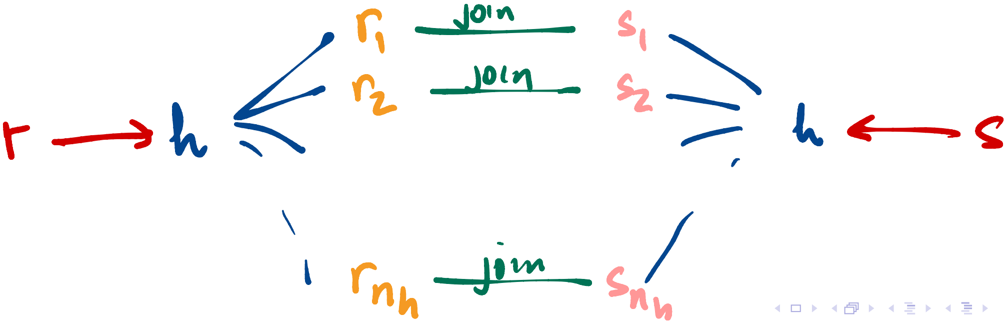
- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + b_s$
  - Block seeks: $\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$
- Hybrid merge join using secondary index

Hash join

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

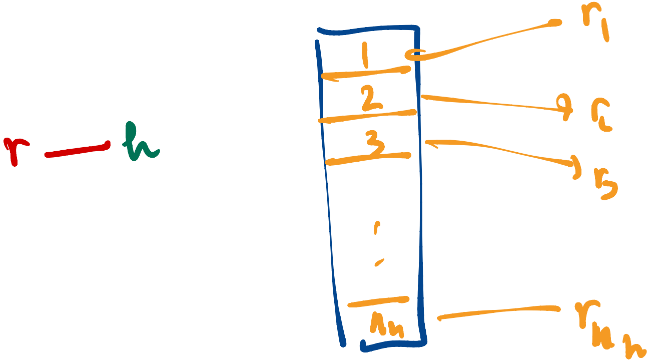
Common attribute A

Apply a hash function to A → n_h output value



Hash join

- (5000 rows, 100 blocks) *Student* ⋈ *Takes* (10000 rows, 400 blocks)



$n_h \leq M$

Assemble hash chunks one block at a time

Query optimization

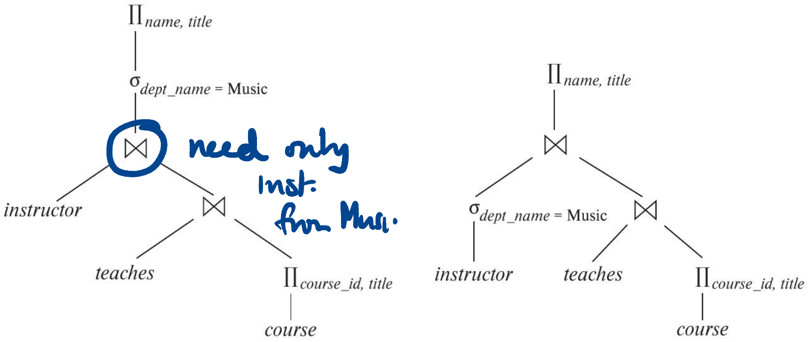
- Choose plan with lowest cost

Query optimization

- Choose plan with lowest cost
- *Find names and course titles of courses taught by instructors from Music Dept*

Query optimization

- Choose plan with lowest cost
- *Find names and course titles of courses taught by instructors from Music Dept*

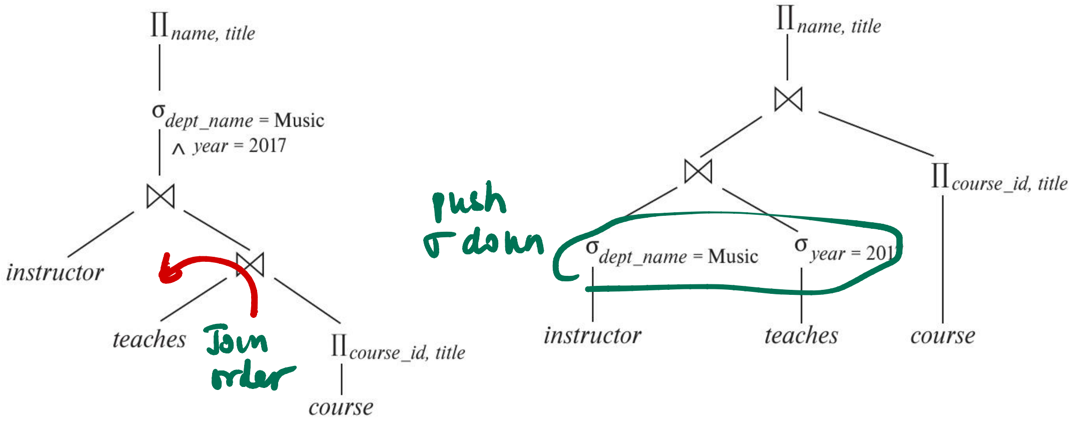

$\Pi_{name,\ title}$
$\sigma_{dept\_name\ =\ \text{Music}}$
$\bowtie$
instructor
$\bowtie$
teaches
$\Pi_{course\_id,\ title}$
course
$\Pi_{name,\ title}$
$\bowtie$
$\sigma_{dept\_name\ =\ \text{Music}}$
$\bowtie$
instructor
teaches
$\Pi_{course\_id,\ title}$
course

need only inst. from Musi.

Transforming expressions

Rules transformation

Transforming expressions



$\Pi_{name,\ title}$
$\sigma_{dept\_name = \text{Music} \wedge year = 2017}$
$\bowtie$
instructor
$\bowtie$
teaches
$\Pi_{course\_id,\ title}$
course
$\Pi_{name,\ title}$
$\bowtie$
$\bowtie$
$\Pi_{course\_id,\ title}$
$\sigma_{dept\_name = \text{Music}}$
$\sigma_{year = 201}$
instructor
teaches
course

Join order

push σ down

Maintaining a database catalogue

- n_r — number of tuples in r
- b_r — number of blocks used by r
- ℓ_r — size of a tuple in r
- f_r — blocking factor of r, how many tuples fit in a block
- $V(A, r)$ — number of distinct values of attribute A in r
  - Store distribution of values as histogram

Join ordering

$$r_1 \bowtie r_2 \bowtie r_3$$

$$(r_1 \bowtie r_2) \bowtie r_3$$

$$r_1 \bowtie (r_2 \bowtie r_3)$$

Heuristics

- Perform selection early

Heuristics

- Perform selection early
- Perform projection early

Heuristics

- Perform selection early
- Perform projection early
- Perform most restrictive selection/join first