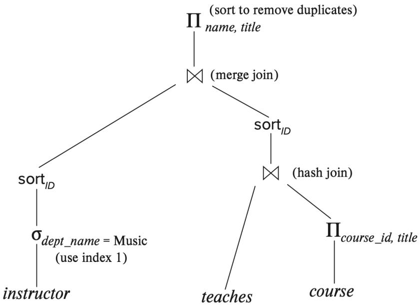
Query Processing (I)

April 28, 2023

DBMS: Operator execution

Purpose:
Execute a dataflow by operation on tuples and files.



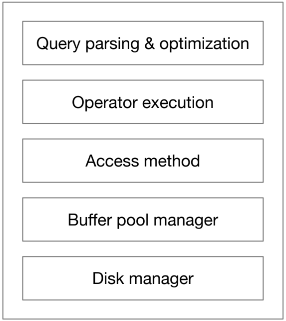
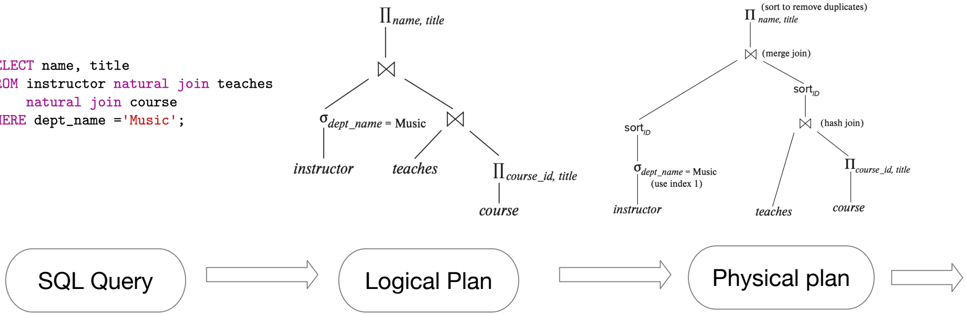


Figure: DBMS architecture

Query processing overview

```sql
SELECT name, title
FROM instructor natural join teaches
     natural join course
WHERE dept_name ='Music';
```



- Each node of a logical plan is a relational operator.
- Each node of a physical plan represents an operator algorithm.
- Data flows from the leaves of the physical plan tree up towards the root.

Notations

- Tables: R, S
- Tuples: t_r, t_s
- Number of tuples: $|R|$, $|S|$
- Number of pages: $P(R)$, $P(S)$
- Number of available buffer pool pages: B
- Cost metric: number of I/O's

Sequential scan

- Scan table R sequentially and process the query
  - Selection over R
  - Projection of R without duplicate elimination
- I/O cost: $P(R)$
- Not counting the cost of writing the result out.
  - Maybe not needed – results may be pipelined into another operator.
  - Same for any algorithm discussed later.

Sorting

Why sorting

- Tuples in a table have no specific order.
- Query may require output be sorted.
  – E.g., `SELECT * from student ORDER BY credit DESC;`
- Several relational operators can be implemented efficiently with sorting.
  – E.g., duplication elimination, aggregation, merge join, set operations.
- External sorting is required when data cannot fit in memory.

External merge sorting

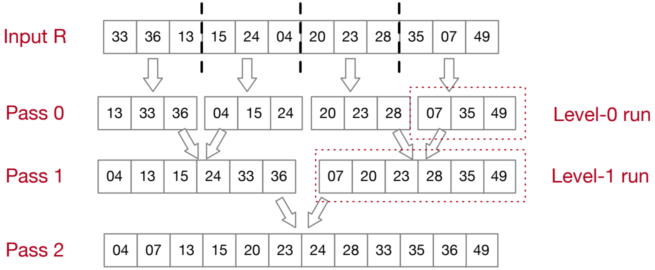
A divide-and-conquer approach to sort a large relation R that cannot fit in memory.

Recall that we have B pages available in the buffer pool.

- Pass 0: read B pages of R each time, sort them, and write out a level-0 run.
- Pass 1: merge $B - 1$ level-0 runs each time, and write out a level-1 run.
- Pass 2: merge $B - 1$ level-1 runs each time, and write out a level-2 run.
- ...
- Final pass produces one sorted run.

External merge example

- $B = 3$, i.e., 3 pages available in buffer pool.
- Each page hols only one tuple.
- In pass 0, all 3 pages are used for sorting.
- In pass i, where $i \geqslant 1$, 2 pages are used for input, and 1 page for output.



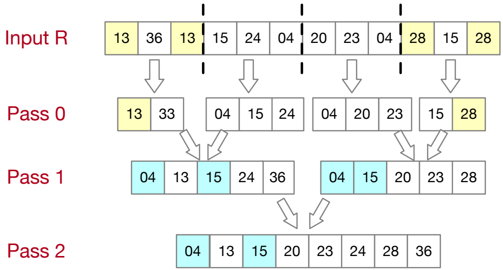
Cost analysis

#(Passes)	$\log_{B-1}\lceil P(R)/B \rceil + 1$
#(Read Pages)	$P(R) * (\log_{B-1}\lceil P(R)/B \rceil + 1)$
#(Write Pages)	$P(R) * \log_{B-1}\lceil P(R)/B \rceil$
Total cost	$2P(R) * \log_{B-1}\lceil P(R)/B \rceil + P(R)$

- Pass 0: read B pages of R each time, sort them, and write out a level-0 run.
- Pass i: merge $(B-1)$ level-$(i-1)$ runs each time, and write out a level-i run.
- Each pass read the entire relation and write it once.
- We do not include the output cost of the final pass as we have discussed.

Sort-based duplication elimination

1. Perform external merge sort.
2. Eliminate duplicates during sort and merge.
3. Cost: same cost as sorting.



Sort-based aggregation

- Sort the tuples on the GROUP BY attributes
- Perform a sequential scan over the sorted data to compute the aggregation.
  – This can be fused into the final pass of sorting.
- Apply partial aggregation on the fly.
- The output will be sorted on the attributes.
- Cost: same cost as sorting.

```
SELECT dept_name, AVG(salary)
FROM instructor
GROUP BY dept_name
```

Agg	Running value
MIN	min
MAX	max
COUNT	count
SUM	sum
AVG	(count, sum)

Join

Naive nested loop join

```
for each tuple t_r in R do
    for each tuple t_s in S do
        if θ(t_r, t_s) then
            add t_r ⋈ t_s to the result
```

Figure: Algorithm for $R \bowtie_\theta S$

- The most basic join algorithm to compute join $R \bowtie_\theta S$.
- R: the outer table, S: the inner table.
- Require no indices and can be used with any kind of join conditions.

Cost analysis

- $P(R) + P(S)$: the buffer pool can hold both tables as input.
- $P(R) + |R| * P(S)$: $B = 3$. Use two buffer pool pages for input, and one page for output.

Example

A	B
10	a
20	b
20	c
40	d

R

A	C
50	e
20	f
20	g
30	h
40	i
50	j

S

A	B	C
20	b	f
20	b	g
20	c	f
20	c	g
40	d	i

$R \bowtie S$

- $|R| = 4,\ |S| = 6,\ P(R) = 2,\ P(S) = 3.$
- If $B = 3$ and R is the outer table, then #(I/O) = 14.
- If $B = 3$ and S is the outer table, then #(I/O) = 15.

Blocked nested loop join

1. for each page P_r in R do
2.   for each page P_s in S do
3.   for each tuple t_r in P_r do
4.   for each tuple t_s in P_s do
5.   if $\theta(t_r, t_s)$ then
6.   add $t_r \bowtie t_s$ to the result

Figure: Improved algorithm for $R \bowtie_\theta S$

- Naive nested loop join is costly since for every tuple in the outer table R, we must do a sequentially scan of the inner table S.
- To maximize the utilization of buffer pool, process tables on a per-page basis, rather than on a per-tuple basis.

Example

A	B
10	a
20	b
20	c
40	d

R

A	C
50	e
20	f
20	g
30	h
40	i
50	j

S

A	B	C
20	b	f
20	b	g
20	c	f
20	c	g
40	d	i

$R \bowtie S$

- $|R| = 4$, $|S| = 6$
- $P(R) = 2$, $P(S) = 3$.
- If $B = 3$ and R is the outer table, then $\#(I/O) = 8$.
- If $B = 3$ and S is the outer table, then $\#(I/O) = 9$.

Cost analysis

- $P(R) + P(R) * P(S)$:use two buffer pool pages for input and another page for output.

In general, if B pages are available in the buffer pool for the join operation, then

- Use $B - 2$ pages to scan the outer table R
- Use one page for inner table scan
- The rest page for buffering the output
- Total cost: $P(R) + \lceil P(R)/(B - 2) \rceil * P(S)$

Merge join

- Require equality predicate, e.g., equi-joins or natural joins.
- If R or S is not sorted by the join attributes, then sort it first.
- All tuples with the same value on the joined attributes are in consecutive order.
- Merge scan the sorted tables and emit tuples that match.

Merge join

```
/* ps/pr points to the first tuple of R/S */
while pr! = EOF & ps! = EOF do
  while t_pr[A] < t_ps[A] do ++pr;
  while t_pr[A] > t_ps[A] do ++ps;
  while t_pr[A] = t_ps[A] do
    pss := ps; /* set pss to the first match */
    while t_pr[A] = t_pss[A] do
      add t_pr ⋈ t_pss to result;
      ++pss;
    ++pr;
  ps := pss; /* all matches processed, advance ps */
```

pr

A	B
10	a
20	b
20	c
40	d

R

ps

pss

A	C
5	e
20	f
20	g
30	h
40	i
50	j

S

A	B	C
20	b	f
20	b	g
20	c	f
20	c	g
40	d	i

R ⋈ S

Cost analysis

- Most cases: Sorting + $P(R) + P(S)$.
- Assumption: Every set of match candidates in S can fit in buffer pool.

- Worst case: Sorting + $P(R) + P(R) * P(S)$
- Assumption: Everything joins and $B = 3$.

Sort-based set operations

- $R \cup S$, $R \cap S$, $R - S$ requires duplication elimination by default.
- Sort R and S in the same order.
- Scan the sorted R and S to produce the desired results in a similar way as in merge join.
- Both R and S require only one pass of scan.
- Cost: sorting + $P(R) + P(S)$