RDBMS and SQL
Query Processing and Optimization

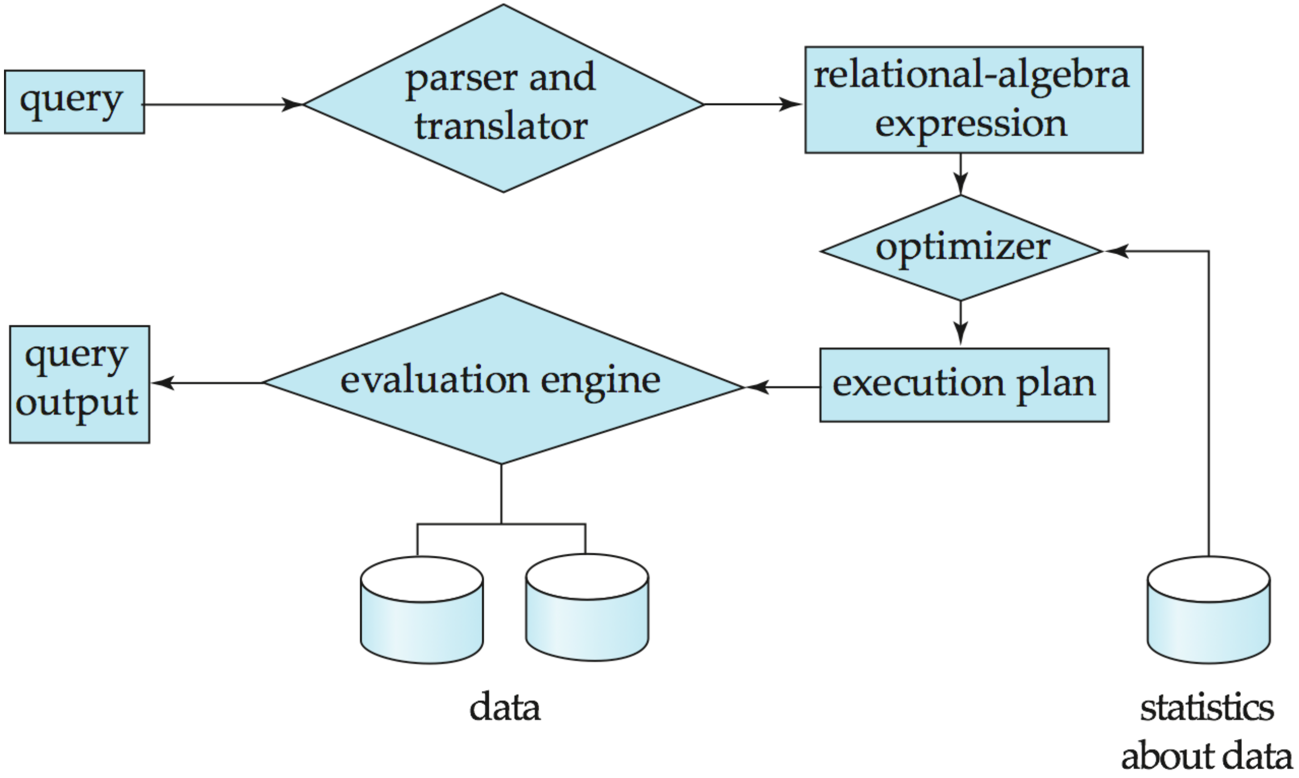
Venkatesh Vinayakarao
venkateshv@cmi.ac.in
http://vvtesh.co.in

Chennai Mathematical Institute

Slide contents are borrowed from the course text. For the authors' original version of slides, visit: https://www.db-book.com/db6/slide-dir/index.html.

Query Processing

How to effectively execute the query?



Optimization

- A relational algebra expression may have many equivalent expressions:
  - $\sigma_{\mathit{salary}<75000}(\Pi_{\mathit{salary}}(\mathit{instructor}))$ is equivalent to $\Pi_{\mathit{salary}}(\sigma_{\mathit{salary}<75000}(\mathit{instructor}))$
- We are interested in finding efficient ways to process the query.

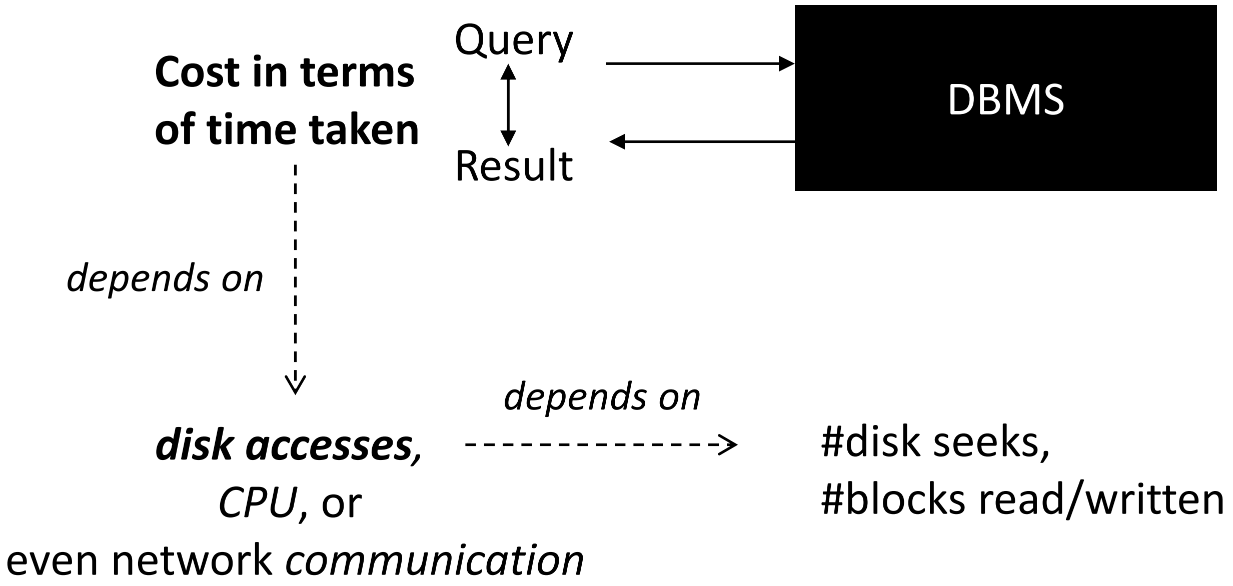
Processing $\sigma_{salary<75000}$(*instructor)*

- $\sigma_{salary<75000}$(*instructor) can be implemented in two ways:*
    - *File Scan-based methods (can be sort-based)*
        - perform complete relation scan and discard instructors with salary $\geq$ 75000
    - *Index-based methods*
        - can use an index on *salary* to find instructors with salary < 75000

Query Cost

- Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
    - *disk accesses, CPU*, or even network *communication*
- Typically disk access is the predominant cost and is also relatively easy to estimate. Measured by considering
  - Number of seeks
  - Number of blocks read
  - Number of blocks written

Query Cost



Cost in terms of time taken

Query

Result

DBMS

depends on

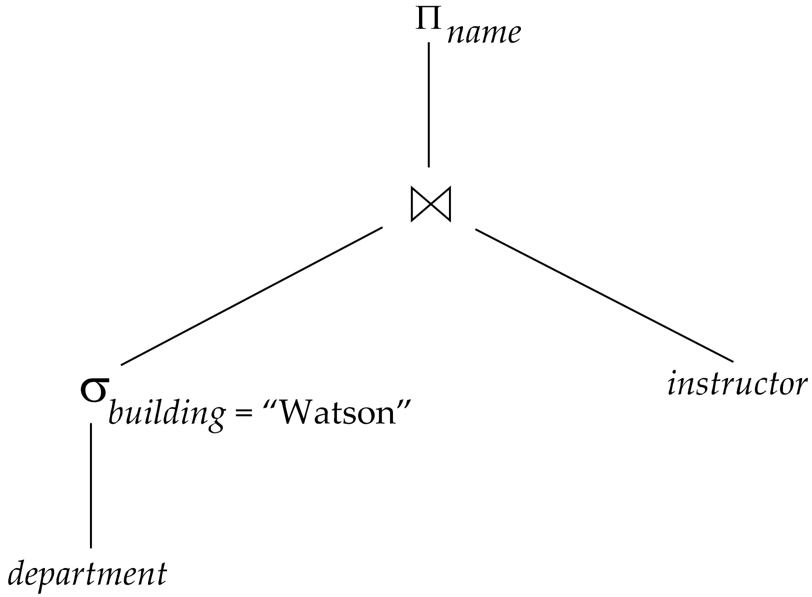
disk accesses,
CPU, or
even network *communication*

depends on

#disk seeks,
#blocks read/written

Query Evaluation

- Query Execution Plan (or Query Evaluation Plan)
  - A sequence of primitive operations to evaluate a query

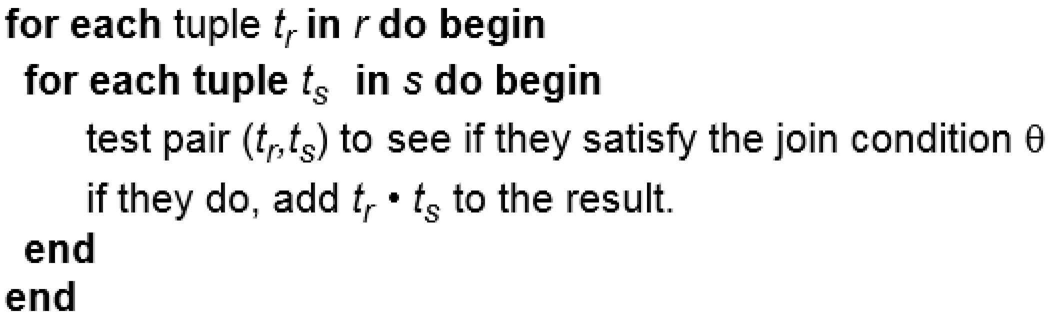


Processing Joins

- Say, we need to perform $r \bowtie_{\theta} s$
- *Assume*
  - *For customer relation*
    - *#records,* $n_{customer} = 10{,}000$
    - *#blocks,* $b_{customer} = 400$
  - *For depositor relation*
    - *#records,* $n_{depositor} = 5{,}000$
    - *#blocks,* $b_{depositor} = 100$

Nested Loop Join

- Algorithm



```
for each tuple $t_r$ in $r$ do begin
 for each tuple $t_s$  in $s$ do begin
      test pair ($t_r$,$t_s$) to see if they satisfy the join condition $\theta$
      if they do, add $t_r \cdot t_s$ to the result.
 end
end
```

- *r* is called the **outer relation** and *s* the **inner relation** of the join.
- Expensive since it examines every pair of tuples in the two relations.
  - Needs n_r * n_s records to be accessed.

Nested Loop Join

- Needs n_r * n_s records to be accessed.
  - If only one block can be retained in memory,
    - Total blocks of s accessed is n_r * $b_{s.}$
    - Total block access for outer relation is b_r.
    - Total block accesses = n_r * b_s + b_r
  - If inner relation can be held in memory
    - Total blocks of s accessed is $b_{s.}$
    - Total block access for outer relation is b_r.
    - Total block accesses = b_s + b_r
- Notice that it is beneficial to keep smaller relation as inner relation.
  - so that (if it fits into memory), it is read only once.

Quiz

- If we had customer as outer relation, what would be the worst-case cost ?

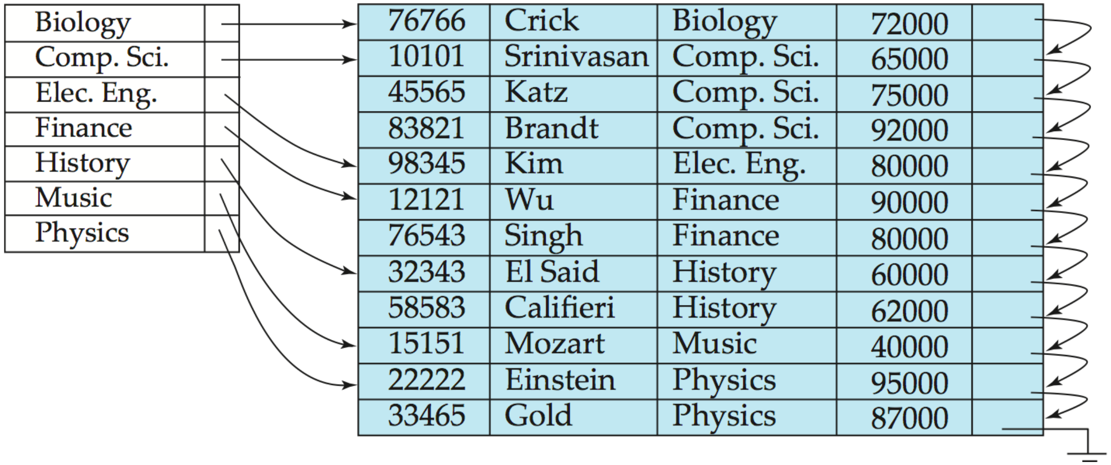
Quiz

- If we had customer as outer relation, what would be the worst-case cost ?
  - 10000 * 100 + 400 = 1,000,400

Indexed Nested-Loop Join

- Assume, you had an index available on the join attribute.
  - index look-ups can replace file scans.

join condition can be seen as select on the index



Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
History	
Music	
Physics	

76766	Crick	Biology	72000	
10101	Srinivasan	Comp. Sci.	65000	
45565	Katz	Comp. Sci.	75000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
12121	Wu	Finance	90000	
76543	Singh	Finance	80000	
32343	El Said	History	60000	
58583	Califieri	History	62000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
33465	Gold	Physics	87000	

Cost of Indexed Nested-Loop Join

- Worst-case assumption: Only one block of r and one block of index could be held in memory.
- For each tuple in r, we perform index lookup over s.
- Cost of join = b_r + n_r * c, where c is the cost of single selection on s.

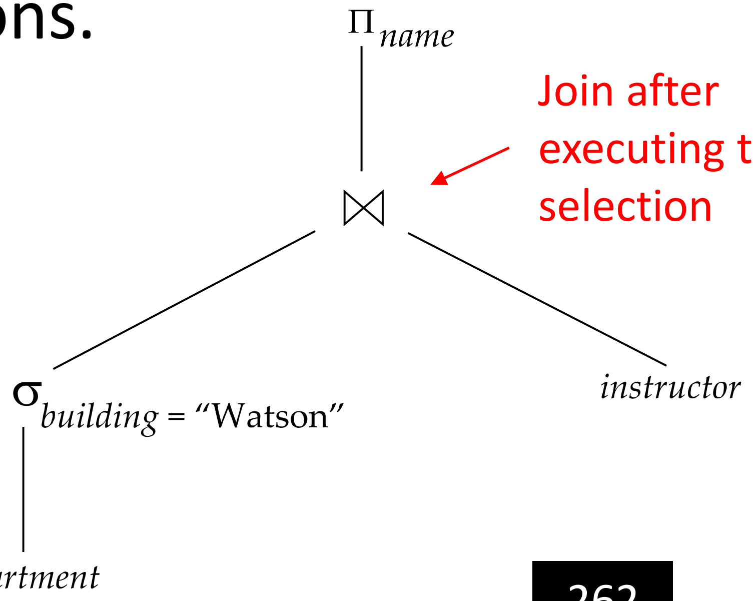
Example

- Outer relation is depositor.
- Inner relation is customer.
- Suppose, customer has a primary B+ tree index on customer-name.
- Avg #entries in each index node of B+ tree = 20
- $n_{customer}$ = 10000, $n_{depositor}$ = 5000, $b_{customer}$ = 100
- height of the tree = 4
- Total cost = 100 + 5000 * (4+1) = 25,100 block accesses.

↑ one more comparison required

Materialized Evaluation

- Evaluate one operation at a time, starting at the lowest-level.
- Use intermediate results materialized into temporary relations to evaluate next-level operations.

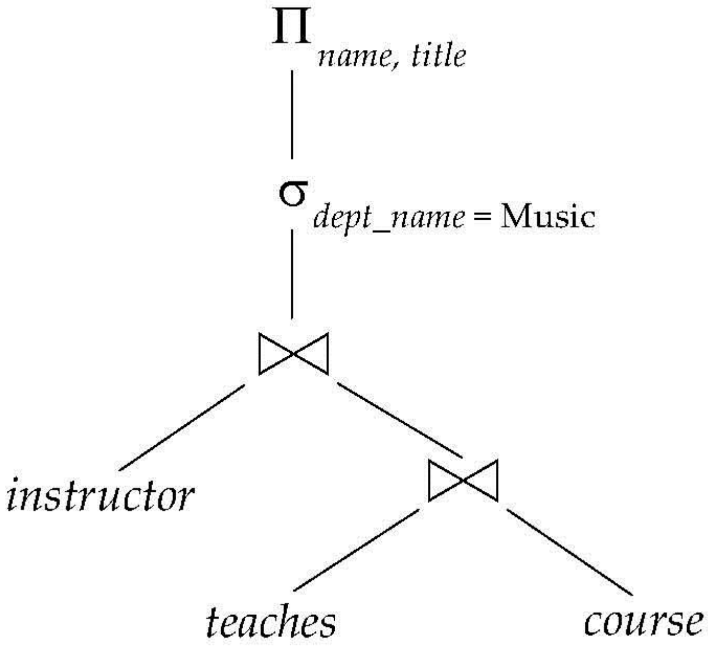


Join after executing the selection

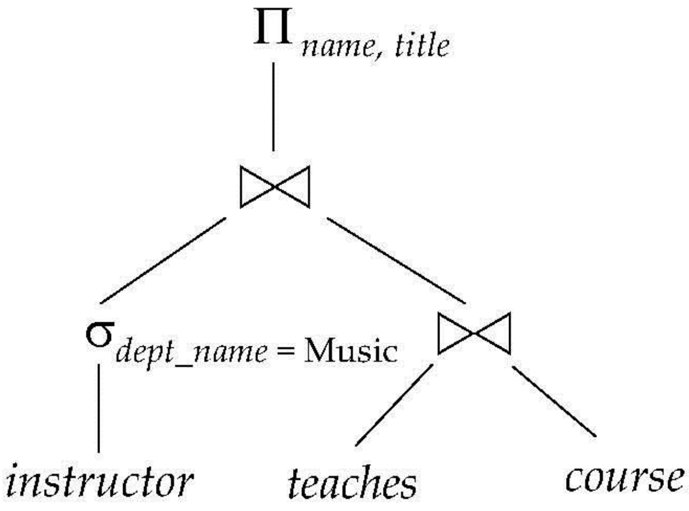
Compute this first

Optimization

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation



Push select down to optimize!



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