Database Technology

Topic 12: Query Processing and Optimization

Olaf Hartig
olaf.hartig@liu.se

LINKÖPING UNIVERSITY

Outline

1. Query representations

- Logical plans
- Physical plans

2. Query processing steps

3. Examples for physical operators

- Table scan
- Sorting
- Duplicate elimination
- Nested loop join
- Sort-merge join
- Hash join

Query Representations

Query Representations: Overview

- While it is processed by a DBMS, a query goes through multiple representations
- Usually, these are:
  1. An expression in the query language (e.g., SQL)
  2. Parse tree / abstract syntax tree (AST)
  3. Logical plan
  4. Physical plan
  5. Compiled program

Logical Plans

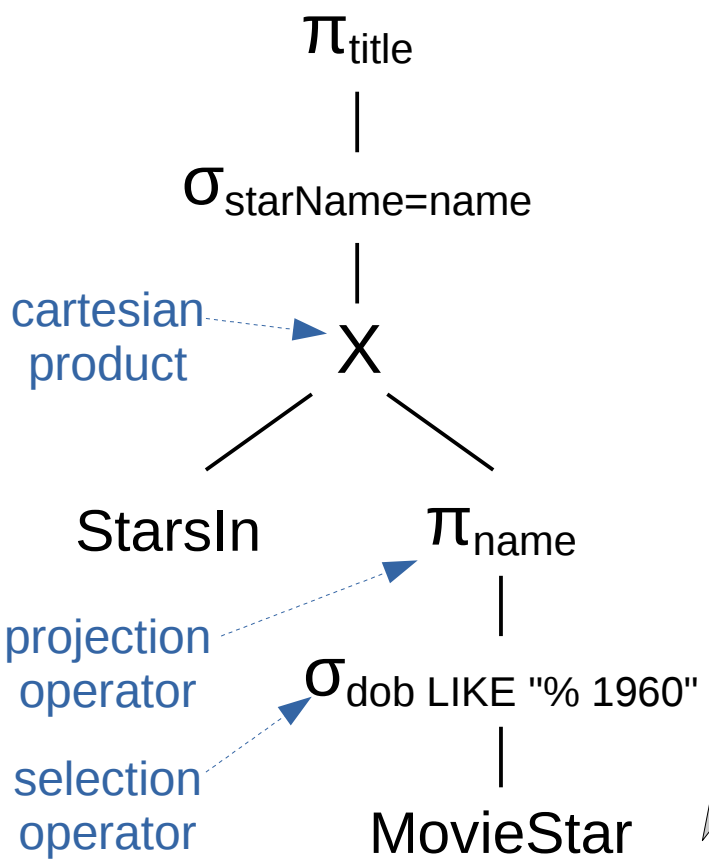
- Represented as an expression in a logical algebra (such as the *relational algebra*)
  - Closely related to the logical data model

- Set of operations for the relational data model
- Similar to the algebra on numbers
  - Operands and results are relations instead of numbers

```
SELECT title
FROM StarsIn
WHERE starName IN (
    SELECT name
    FROM MovieStar
    WHERE dob LIKE "% 1960" )
```

$$\pi_{title}(\sigma_{starName=name}(StarsIn \times \pi_{name}(\sigma_{dob\ LIKE\ "\%\ 1960"}(MovieStar))))$$

Logical Plans



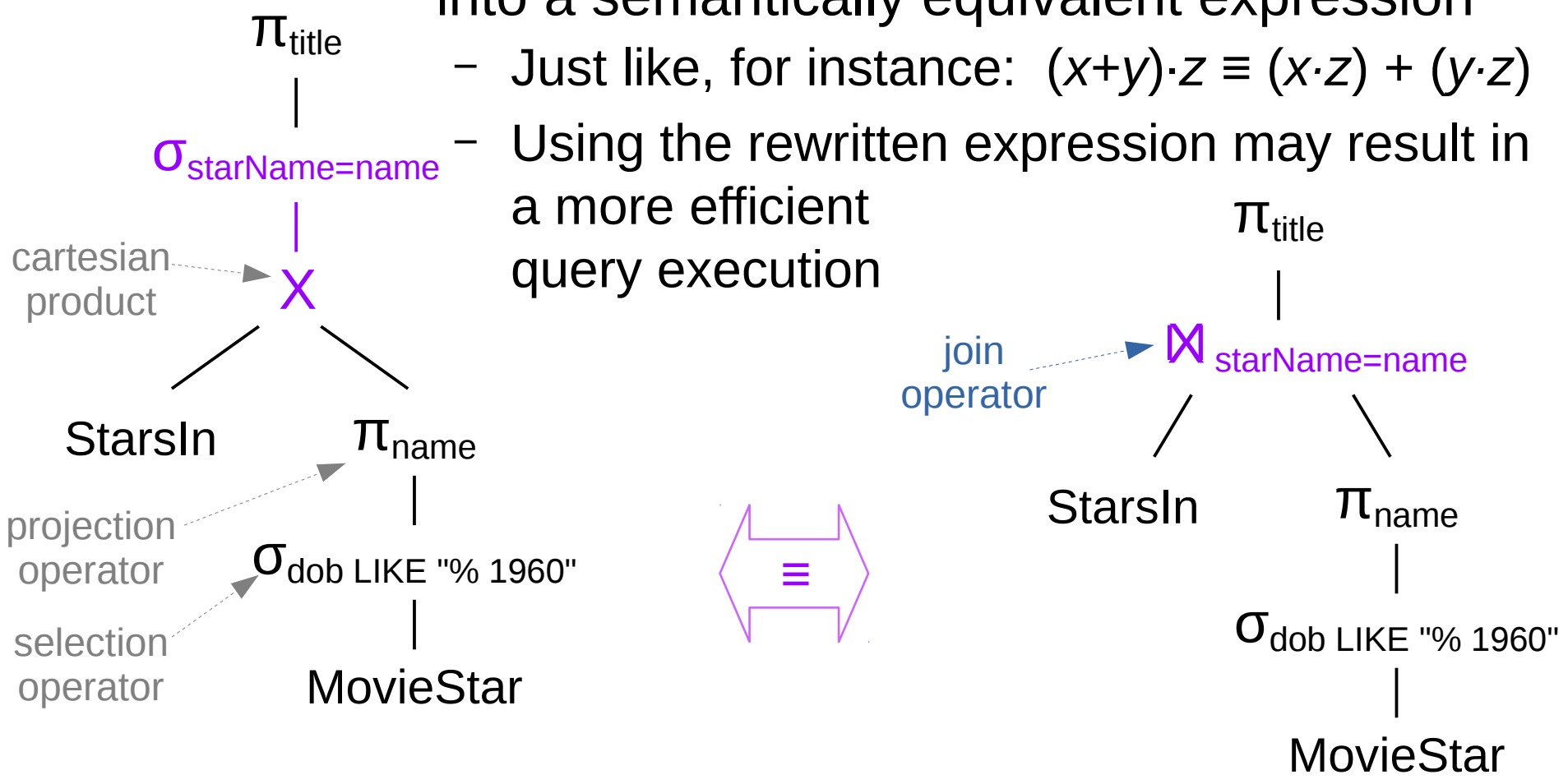
- Represented as an expression in a logical algebra (such as the *relational algebra*)
  - Closely related to the logical data model
- Can be visualized as a tree of logical operators

```
SELECT title
FROM StarsIn
WHERE starName IN (
    SELECT name
    FROM MovieStar
    WHERE dob LIKE "% 1960" )
```

$$\pi_{title}(\sigma_{starName=name}(StarsIn\ X\ \pi_{name}(\sigma_{dob\ LIKE\ "\%\ 1960"}(MovieStar))))$$

Logical Optimization

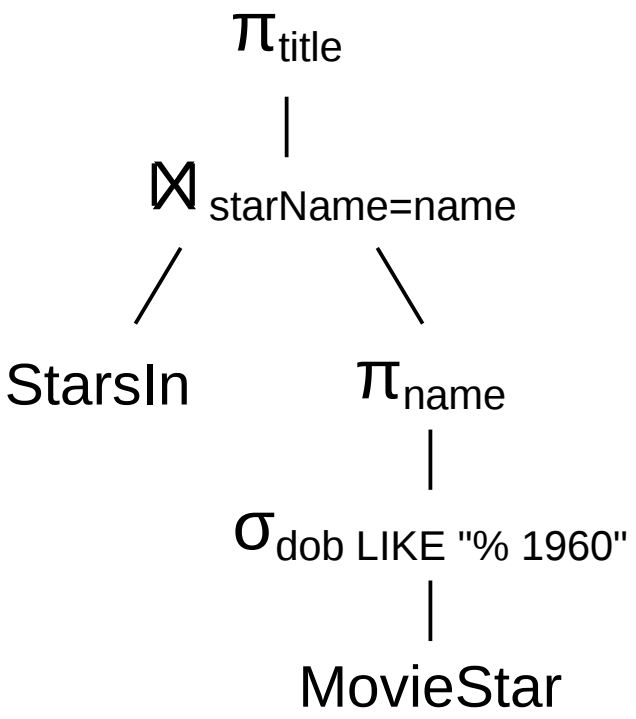
- An algebra expression may be rewritten into a semantically equivalent expression
  - Just like, for instance: $(x+y)\cdot z \equiv (x\cdot z) + (y\cdot z)$
  - Using the rewritten expression may result in a more efficient query execution



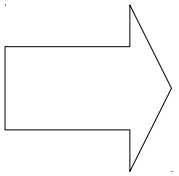
Physical Plans

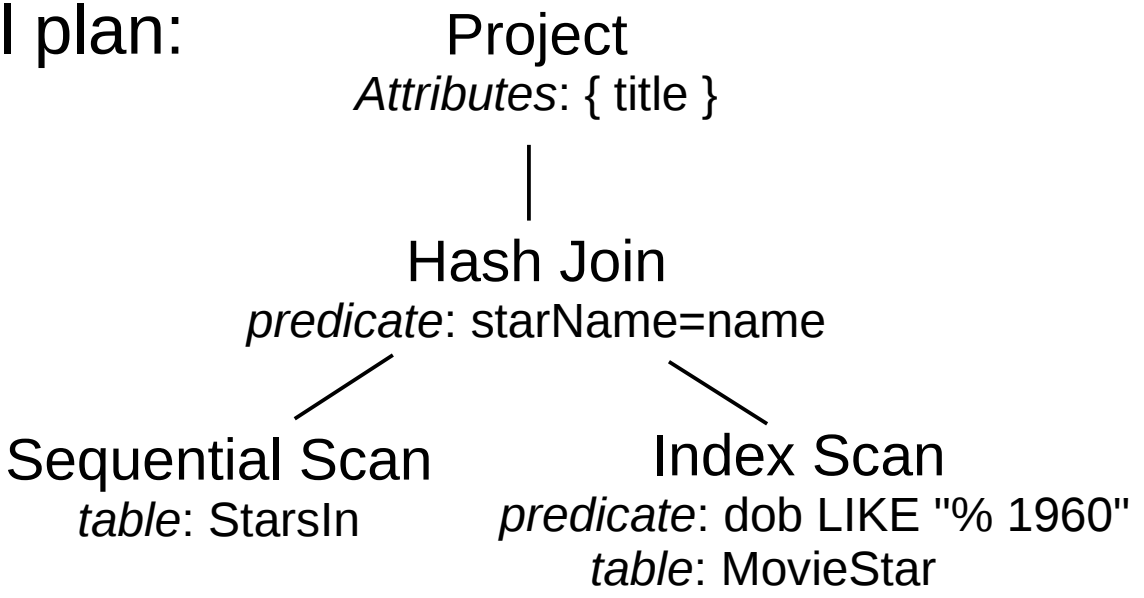
- Also often called *query execution plan* (QEP)
- Represented as an expression in a physical algebra
  - Algebra over sequences of tuples
  - Operators are called physical operators
- Physical operators come with a specific algorithm
  - e.g., a nested loops join (see later)
- Physical operators are associated with a cost function
  - Can be used to calculate the amount of resources needed for their execution
  - Typically, focus on I/O cost only

Physical Plans (Example)



Possible physical plan for the given logical plan:



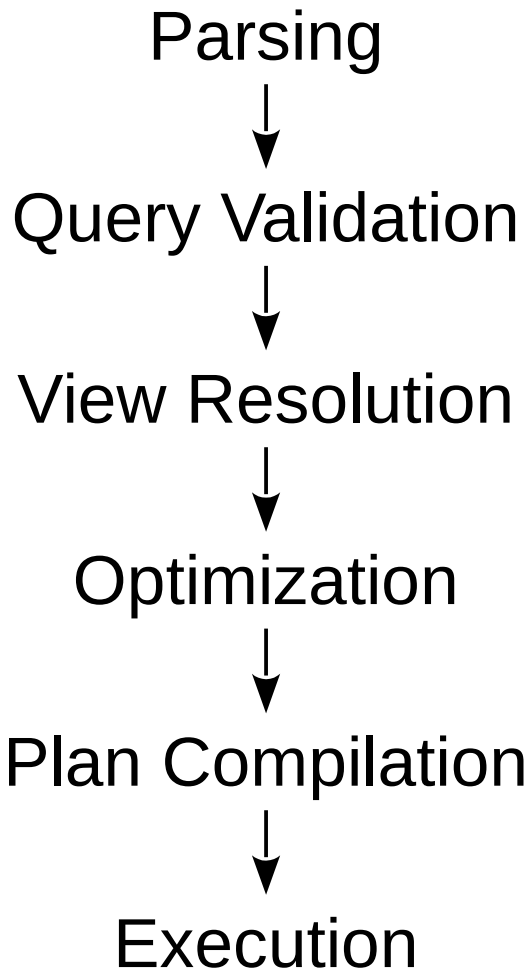


Logical vs. Physical Operators

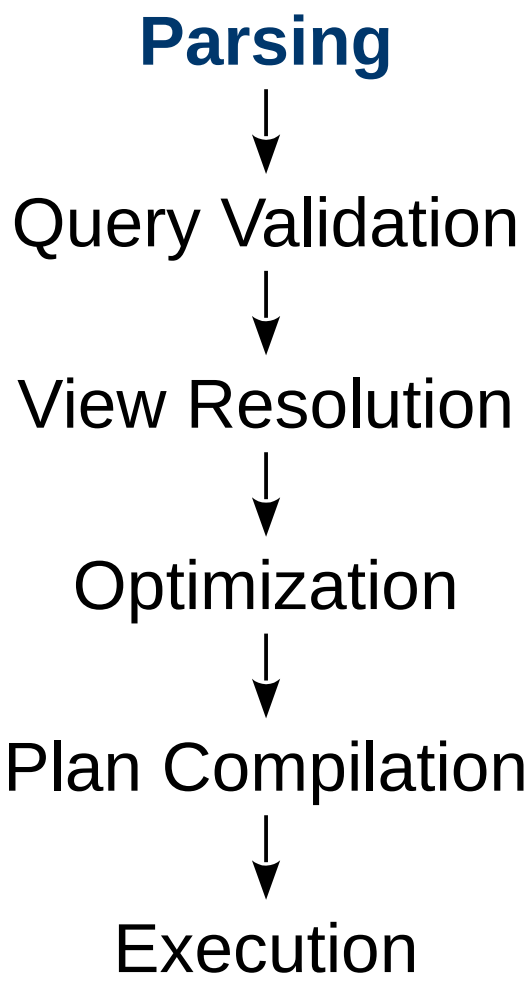
- Logical operators and physical operators do not necessarily map directly into one another
  - For instance,
    - … most join algorithms can project out attributes (without duplicate elimination)
    - … a (physical) duplicate-removal operator implements only a part of the (logical) projection operator
    - … a (physical) sort operator has no counterpart in a (set-based) logical algebra

Query Processing Steps

Query Processing Steps: Overview

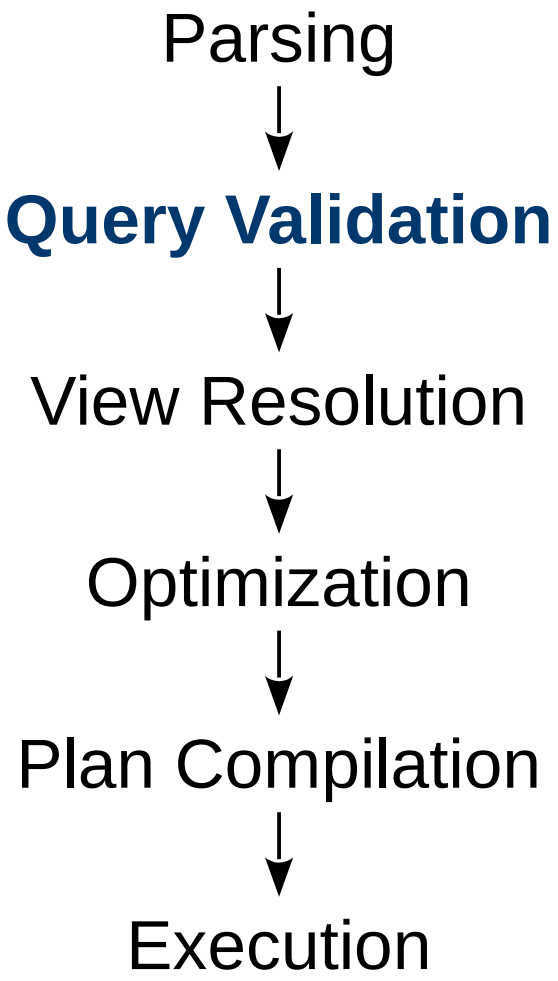


Query Processing Steps: Parsing



- Input: SQL query string
- Output: internal representation (e.g., based on relational algebra)

Query Processing Steps: Validation



Check whether:

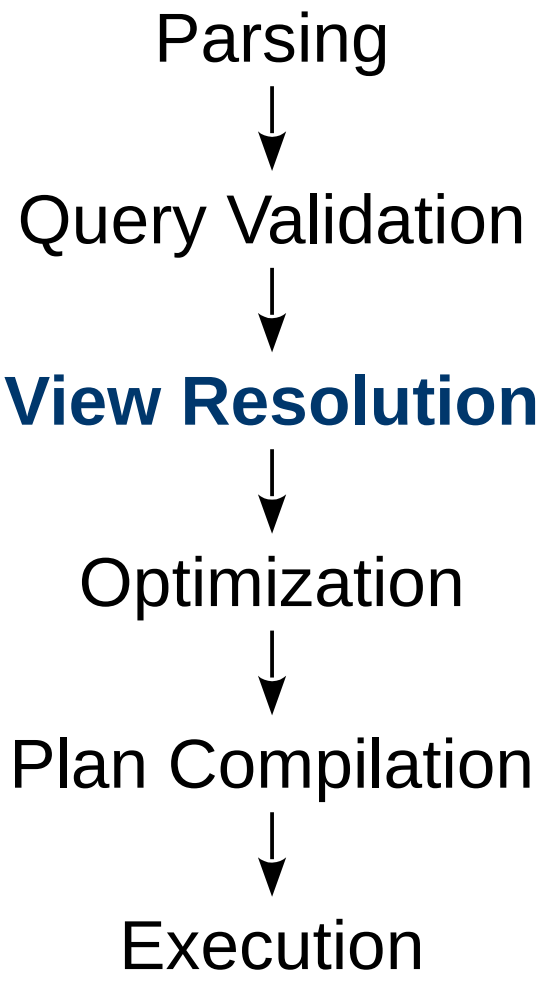
… mentioned tables, attributes, etc., exist

… comparisons are feasible (e.g., comparability of attribute types)

… aggregation queries have a valid SELECT clause

… etc.

Query Processing Steps: Views



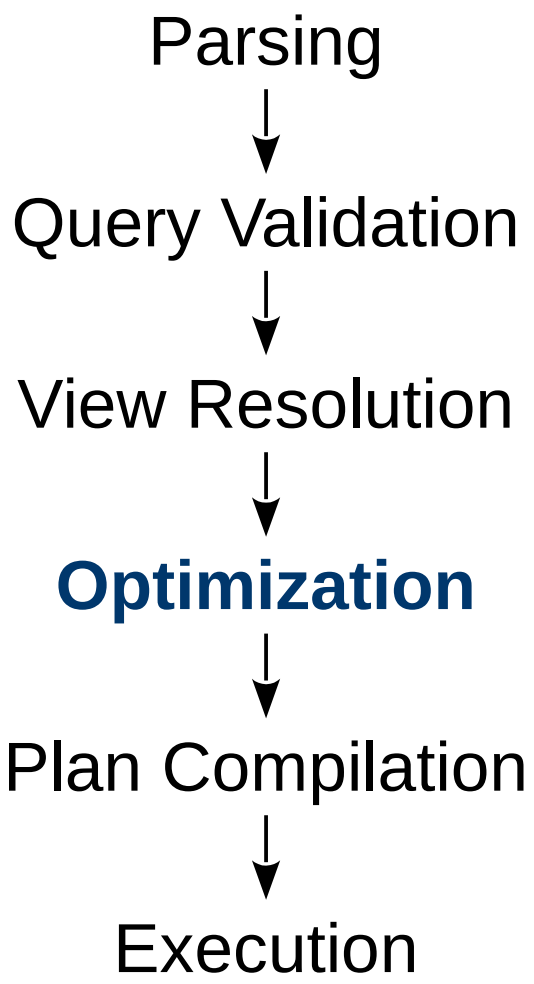
- Substitute each reference to a view by the corresponding view definition
- Example:

```
SELECT …
FROM MyView, …
WHERE … ;
```

CREATE VIEW MyView
AS SELECT … ;

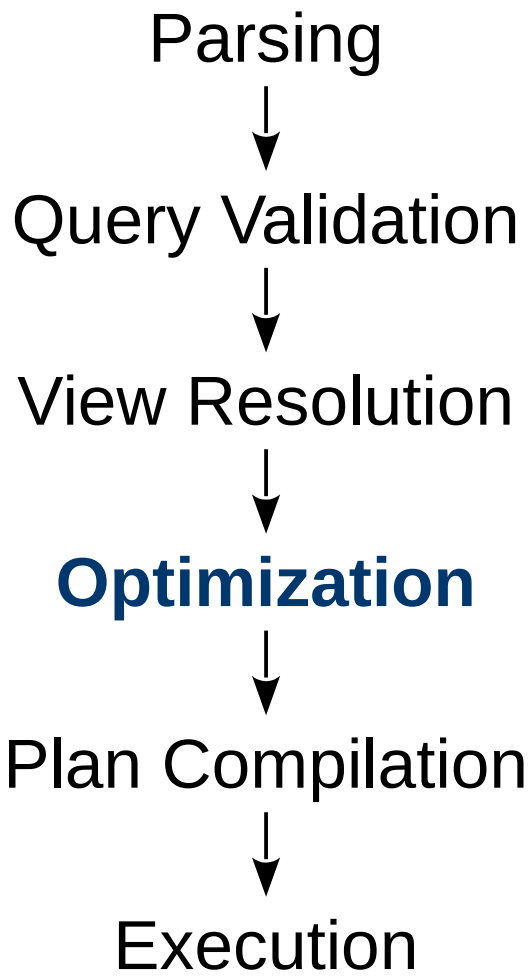
```
SELECT …
FROM (SELECT …) AS MyView, …
WHERE … ;
```

Query Processing Steps: Optimize



- Consider possible *query execution plans* (QEPs)
  - Different QEPs have different costs (i.e., resources needed for their execution)
- Output: an *efficient* QEP
  - i.e., *estimated* cost is the lowest or comparatively low
- This task is all but trivial ...

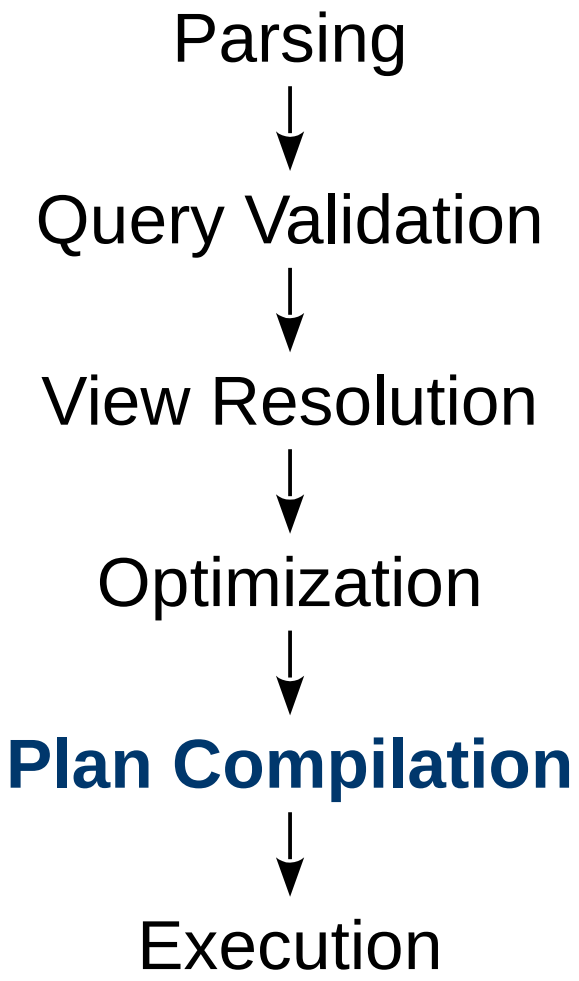
Query Processing Steps: Optimize



… this task is all but trivial!

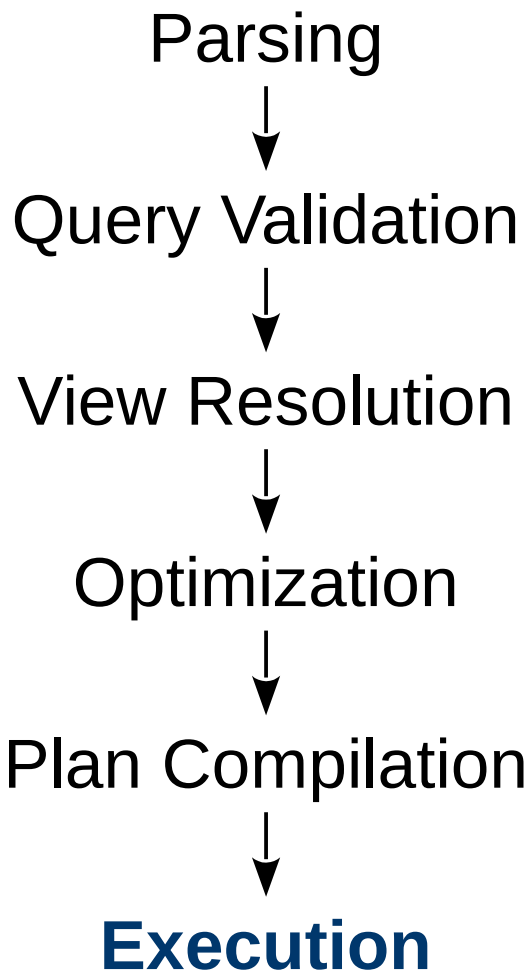
- Space of all possible QEPs is huge
  - Optimizers enumerate only a restricted subset (called: *search space*)
- A desirable optimizer:
  - Search space has low-cost QEPs
  - Enumeration algorithm is efficient
  - Cost estimation is accurate

Query Processing Steps: Compile



- Translate the selected QEP into a representation that is ready for execution
  - e.g., interpreted language, compiled machine code

Query Processing Steps: Execution



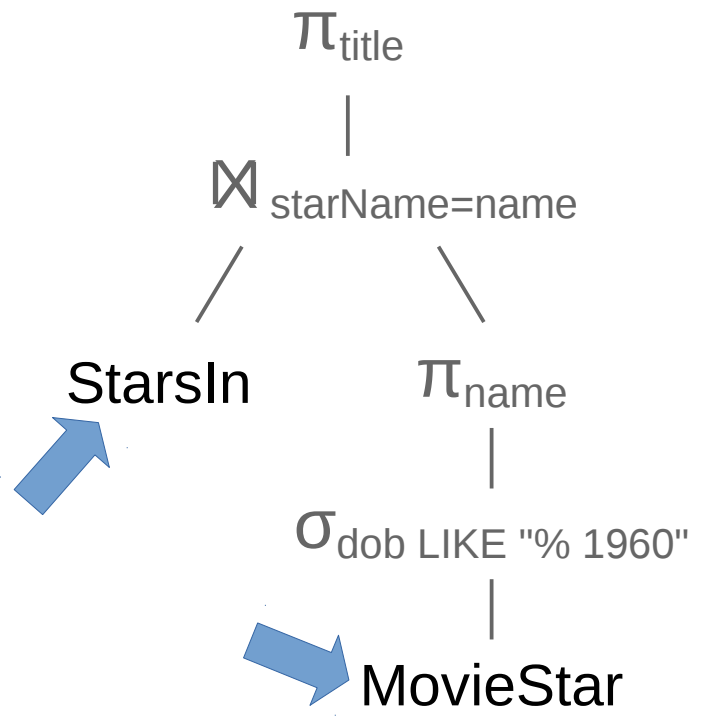
- Execute the compiled plan
- Return the query result via the respective interface

Examples for Physical Operators

- Table Scan
- Sorting
- Duplicate Elimination
- Nested Loop Join
- Sort-Merge Join
- Hash Join

Table Scan

- Leaves of logical operator trees are tables
- Accessing them completely implies a *sequential scan*
  - Load each file block of the table
  - Sequentially scanning a table that occupies *n* blocks has *n* I/O cost



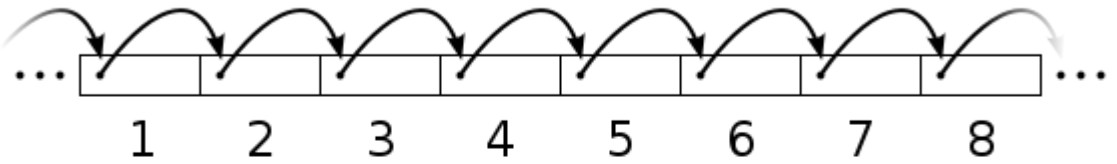


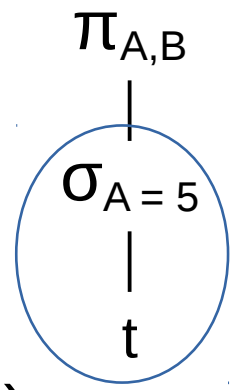
Table Scan (cont'd)

- Combining the scan with the next operation in the plan is often better
- Example:

```
SELECT A, B FROM t WHERE A = 5
```

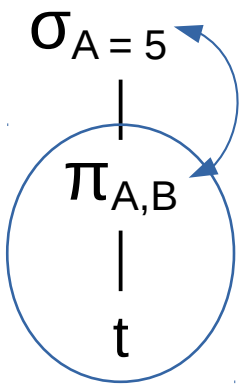
Filtering (also called *selection*): if we have an index on **A**, perform an *index scan* instead (i.e., obtain relevant tuples by accessing the index)

  - Especially effective if **A** is a key



Projection: integrate into the table scan (i.e., read all tuples but only pass on attributes that are needed)

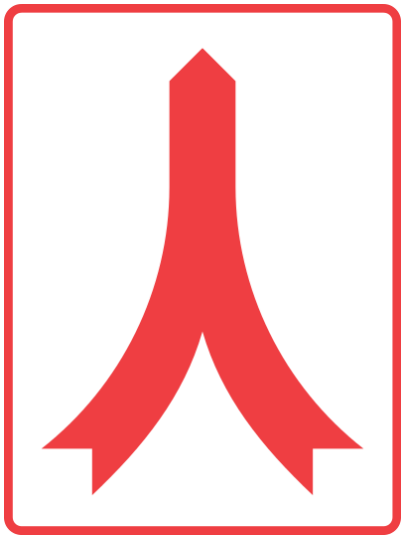
  - Can also be combined with index scan



Sorting

- Many physical operators require input to be sorted
- The (unsorted) input may not fit into main memory
- We need an *external sorting* algorithm
  - Intermediate results are stored temporarily on secondary memory

(Simple) External Merge Sort



- First, sort each file block internally
- Group these sorted blocks into pairs,
  for each pair merge its two blocks
- Next, each of these groups is merged with another group, resulting in groups of four blocks
- And so on …
- The final group is the completely sorted file

- This strategy uses 3 page buffers in main memory
- If more buffers are available, we should exploit them…

External Merge Sort

- Suppose m+1 buffers are available in main memory, and the input occupies p file blocks
- Stage 1:
  - Group blocks into p/m groups and sort each group (using an m-way merge after sorting each of its blocks internally)
- Each additional stage, i.e., n-th stage for all $n > 1$:
  - Group the sorted groups from stage $n–1$ into larger groups such that each larger group contains m previous groups (and m^n file blocks)
  - For each of these new groups, perform an m-way merge
  - If $m^{n+1} > p$, we are done
- Maximum number of stages: $\lceil \log_m(p) \rceil$
- Maximum number of blocks read and written: $2 \times p \times \lceil \log_m(p) \rceil$

Duplicate Elimination (Option 1)

- Two steps:
  1. Sort input table (or intermediate result from previous operator) on DISTINCT column(s)
     - Can be skipped if input is already sorted
  2. Scan sorted table and each tuple only once

- Advantage: Generated output is sorted
- Disadvantage: Cannot be pipelined

Duplicate Elimination (Option 2)

- Idea: scan the input and gradually populate an internal data structure that holds each unique tuple once
- For each input tuple, check if it has already been seen
  No: insert tuple into the data structure and output the tuple
  Yes: skip to the next input tuple
- Possible data structures:
  - Hash table – might be faster, needs good hash function
  - Binary tree – some cost for balancing, robust

- Advantage: can be pipelined
- Disadvantage: does not sort output
  - but existing sorting would remain

Nested Loops Join (NLJ)

- General idea (assuming join condition is *R.A*=*S.A*):

```
FOR EACH tuple r in relation R DO
  FOR EACH tuple s in relation S DO
    IF r.A = s.A THEN output tuple r∪s
```

- Of course, we do this for tables that are distributed over multiple pages that we have to read from disk:

```
FOR EACH page p of relation R DO
  FOR EACH page q of relation S DO
    FOR EACH tuple r on page p DO
      FOR EACH tuple s on page q DO
        IF r.A = s.A THEN output tuple r∪s
```

- I/O cost: pages(*R*) + pages(*R*) X pages(*S*)

Possible Improvements of NLJ

- Block nested loop join
  - For the outer-loop relation, read multiple blocks per iteration (as many as free buffers in main memory)
- Zig-zag join
  - For the inner-loop relation, alternate between loading its blocks forward and backward
- Index nested loop join
  - If there is an index on the join column of one of the two relations, make it the inner relation and use the index instead of scanning the whole data file in every iteration

Sort-Merge Join

- Sort phase
  - Sort *both* inputs on their respective join attributes
  - May need to use an external sorting algorithm
  - Sorting may be skipped for inputs that are already sorted
- Merge phase
  - Synchronously iterate over both (sorted) inputs
  - Merge and output tuples that can be joined
  - Caution if join values may appear multiple times

Sort-Merge Join (Cost Estimation)

- I/O costs for sort phase:

$$\underbrace{2 \times pages(R) \times \lceil \log(pages(R)) \rceil}_{\text{sorting } R} + \underbrace{2 \times pages(S) \times \lceil \log(pages(S)) \rceil}_{\text{sorting } S}$$

- I/O costs for merge phase:

$$pages(R) + pages(S)$$

Hash Join

- Idea: use join attributes as hash keys in both input tables
- Choose hash function for building hash tables with *m* buckets (where *m* is the number of page buffers available in main memory)

Hash Join

- Partitioning phase:
  - Scan relation *R* and populate its hash table
  - Whenever the page buffer for a hash bucket is full, write it to disk and start a new page for that buffer
  - Finally, write the remaining page buffers to disk
  - Do the same for relation *S* (using the same hash function!)
  - Result: hash-partitioned versions of both relations on disk
  - Now, we only need to compare tuples in corresponding partitions
- Probing phase:
  - Read in a complete partition from *R* (assuming |*R*| < |*S*|)
  - Scan the corresponding partition of *S* and produce join tuples
- I/O costs:

$$\underbrace{2\times(pages(R)+pages(S))}_{\text{partitioning}}+\underbrace{(pages(R)+pages(S))}_{\text{probing}}$$

Summary

Summary

- For each query, different physical operators can be combined into different, semantically equivalent QEPs
- Every physical operator comes with an algorithm
- Commonly used techniques for many of these algorithms:
  - *Combining*: multiple tasks may be combined once some input data has been read in
  - *Partitioning*: by sorting or by hashing, we can partition the input(s) and ignore many irrelevant combinations (less work)
  - *Indexing*: existing indexes may be exploited to reduce work to relevant parts of the input
- Each of these algorithms has a specific cost
- Thus, different QEPs have different costs
- Actual cost can only be estimated (w/o executing the QEP)

www.liu.se

LINKÖPING UNIVERSITY