Chapter 5

Query Processing and Optimization

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

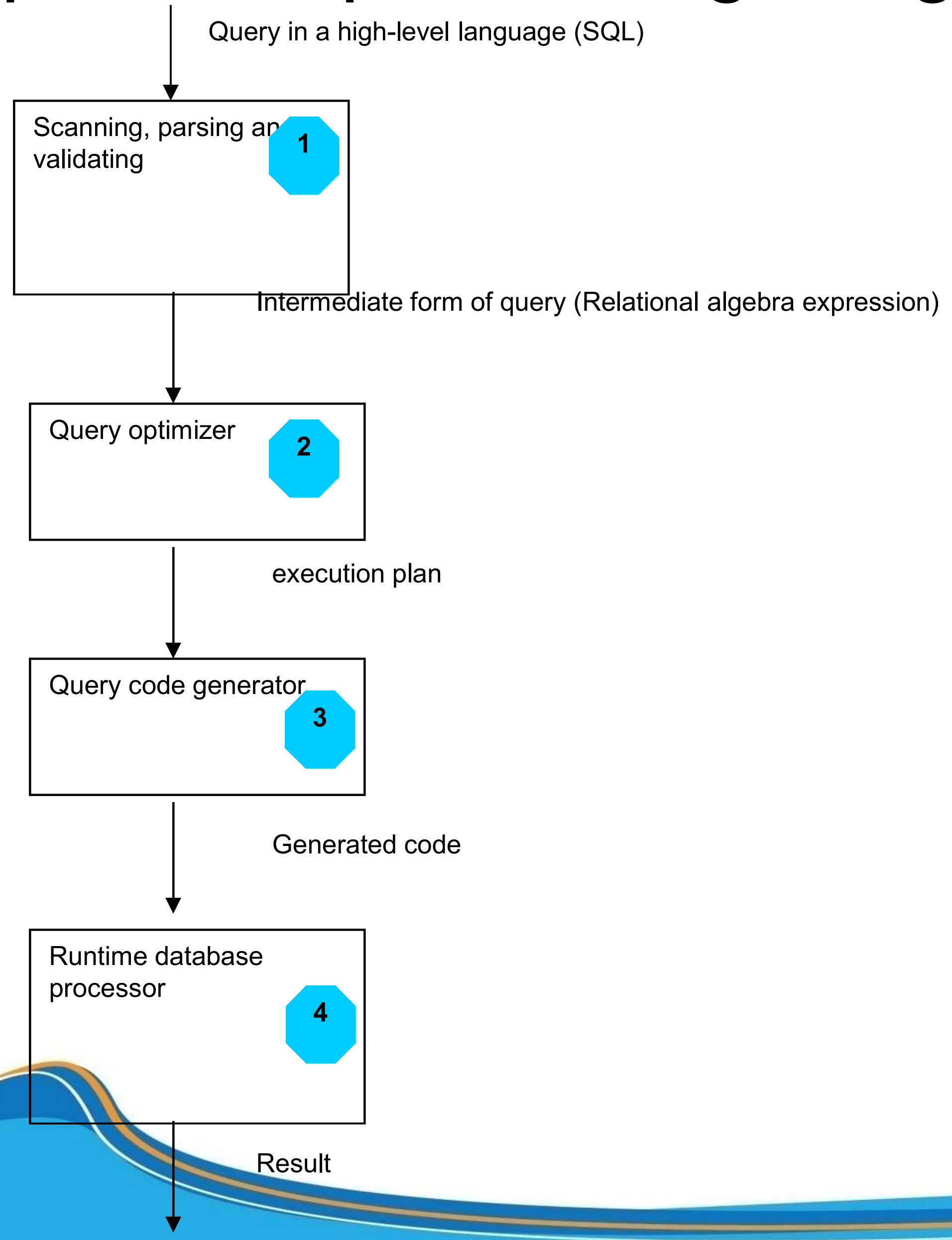
Goal

- A query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as query optimization.

Outline

1. Concepts
2. Database access algorithms
3. Using heuristics in query optimization
4. Using selectivity and cost estimates in query optimization

Steps when processing a high-level query


Query in a high-level language (SQL)
Scanning, parsing and validating
1
Intermediate form of query (Relational algebra expression)
Query optimizer
2
execution plan
Query code generator
3
Generated code
Runtime database processor
4
Result

Steps when processing a query

- Step 1
  - Scan
    - The scanner identifies the languages tokens such as: SQL keywords, attribute names, relation names.
  - Parse
    - The parser checks the query syntax.
  - Validate
    - Checking if all attribute and relation names are valid and semantically meaningful names in the schema.
    - Checking ambiguous attribute names
    - Checking data types in the comparisons.
  - Create an internal representation of the query, using query tree, query graph, ...

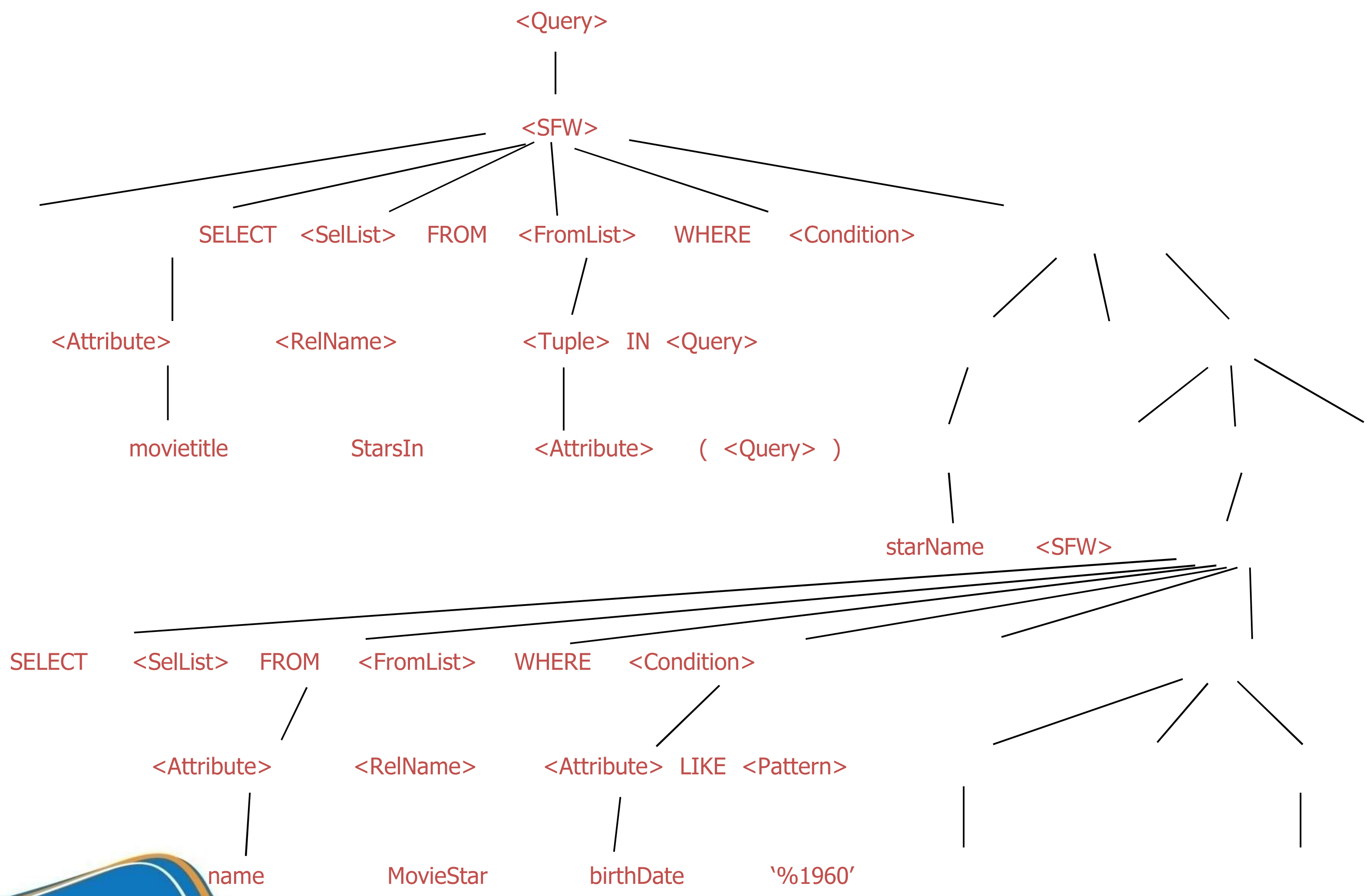
Parse tree

StarsIn (movieTitle, movieYear, starName)
MovieStar (name, address, gender, birthdate)

Find the movie with stars born in 1960

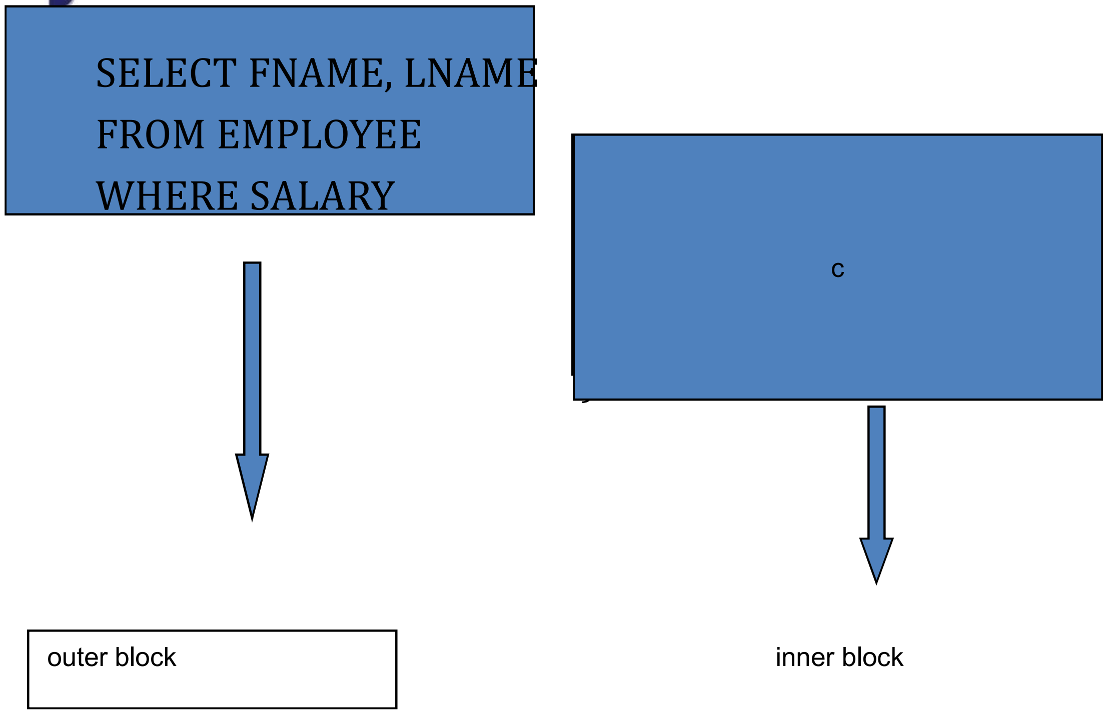
```
SELECT movieTitle
FROM StarsIn
WHERE starName IN (
          SELECT name
          FROM MovieStar
          WHERE birthdate LIKE ‘%1960’);
```

Parse tree


<Query>
<SFW>
SELECT
<SelList>
FROM
<FromList>
WHERE
<Condition>
<Attribute>
<RelName>
<Tuple>
IN
<Query>
movietitle
StarsIn
<Attribute>
(<Query>)
starName
<SFW>
SELECT
<SelList>
FROM
<FromList>
WHERE
<Condition>
<Attribute>
<RelName>
<Attribute>
LIKE
<Pattern>
name
MovieStar
birthDate
'%1960'

Translating SQL queries into relational algebra

- SQL queries are decomposed into query blocks (QB).
  - QB is the basic units that can be translated into the algebraic operators and optimized.
  - A QB contains a single SELEC-FROM-WHERE-GROUP BY-HAVING.
  - Nested queries within a query are identified as separate QB.
  - Aggregate operators (MAX, MIN, SUM) are represented by the extended algebra.



```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE SALARY
```

The QUERY OPTIMIZER would then choose an execution plan for each block.

- Step 2
  - The DBMS then devise an execution strategy for retrieving the result of the query from the database files.
  - A query typically has many possible execution strategies. The process of choosing a suitable one for processing a query is known as query optimization.
- Step 3
  - The query optimizer module has the task of producing an execution plan, and the code generator generates the code to execute that plan.
- Step 4
  - The runtime database processor has the task of running the query code to produce the query result.

- The term optimization is actually a misnomer because in some cases the chosen execution plan is not the optimal (best) strategy – it is just a reasonably efficient strategy for executing the query.
- Finding the optimal strategy is usually to time-comsuming and may require information on how the file are implemented and even on the contents of the files.

- Database Access Algorithms
  - Extenal sorting
  - SELECT algorithm
  - JOIN algorithm
  - SET operations
  - Aggregate operations
  - Outer join algorithm

External sorting

- Sorting is one of the primary algorithm used in query processing. For example, ORDER BY.
- Sorting is also a key component in sort-merge algorithm used in JOIN and other operations: UNION, INTERSECTION, duplicate elimination for the PROJECT operation.
- Sorting should be avoided if an appropriate index exists to allow ordered access to the records.
- External sorting refers to sorting algorithms that are suitable for large files or records stored on disk that do not fit entirely in the main memory.
- Sort-Merge:
  - Consists of two phased: the sorting phase and the merging one.
  - Sorting small subfiles (runs) of the main file and then merge the sorted runs, creating larger sorted runs that are merged in turn.
  - The size of a run and number of initial runs n_R is dictated by the number of blocks (b) on file and the available buffer space n_B.
    - If $n_B = 5$ and b = 1024 blocks then $nR = \lceil b/nB \rceil = 205$ runs. After the sort phase, 205 sorted runs are stored as temporary subfiles on disk.

SELECT operation

- There are many options for executing a SELECT operation; some depend on the file having specific access paths and may apply only to certain types of selection condition:
  - S1: Linear Search (brute force): retrieve every record in the file, and test whether its attribute values satisfy the selection condition.
  - S2: Binary search: if the selection condition involves an equality comparison on the key attribute on which the file is ordered, binary search can be used. OP1: $\sigma_{SSN = '12345'}$ EMP
  - S3: Using a primary index (or hash key): if the selection condition involves an equality comparison on a key attribute with a primary index (or hash key).

- S4: using a primary index to retrieve multiple records: if the comparison condition is >, >=, <, <= on a key field with a primary index.

  OP2: $\sigma_{DNUMBER>5}$ DEP

  Use the index to file the record satisfying the corresponding equality condition (DNUMBER = 5), then retrieve all subsequent records in the ordered file.

- S5: Using a clustering index to retrieve multiple records: if the selectioin condition involves an equality comparison on a non-key attribute with a clustering index:

  OP3: $\sigma_{DNO=5}$ EMP

- S6: Using a secondary index on an equality comparison: this search method can be used to retrieve a single record if the indexing field is a key or to retrieve multiple records if the indexing field is not a key. This can be used in the non-equal comparison.

SELECT operation

- For complex selection
  - Conjunctive selection using an idividual index.
  - Conjunctive selection using a composite index.
  - Conjunctive selection by intersection of record pointers.

JOIN operation R $\bowtie_{A=B}$ S

- J1: Nested-loop join: for each record t in R (outer loop), retrieve everyh record s from S (inner loop) and test whether the two records satisfy the joint condition R[A] = s[B].
- J2: Single-loop join: if the index (or hash key) exists for one of the two join attributes – say, B of S – retrieve each record t in R, one at a time (single loop), and then use the access structure to retrieve directly all matching records from S that satisty S[B] = R[A].
- J3: Sort-merge join: if the records of R and S are physically sorted by values of the join attributes A and B (or we sort them first), both files are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B.
- J4: Hash join (kết băm): the records of files R and S are both hashed to the same hash file, using the same hashing function on the join attributes A of R and B of S as hash key. Both files are then scanned to find the matched records.

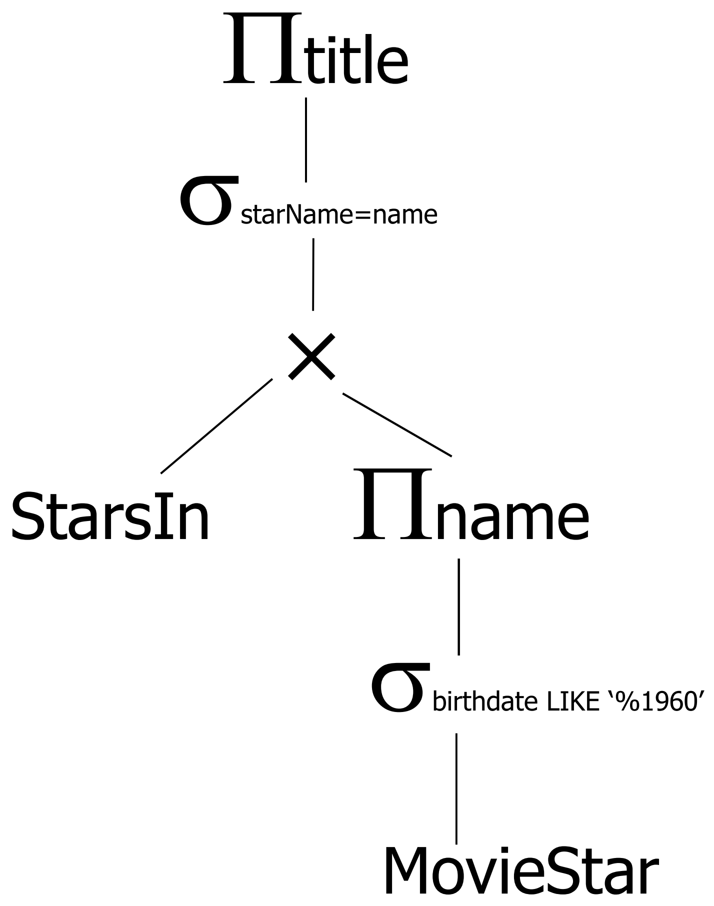
PROJECTION operation $\Pi_{\text{ATT\_LIST}}$ (R)

- If ATT_LIST includes a key of R, the operation is easy to implement.
- If ATT_LIST does not includes a key of R, duplicate tuples must be eliminated.
  - Sorting and eliminating.
  - Hashing can also be used.

SET operations

- UNION, INTERSECT, SET DIFFERENCE:
  - Apply only to union-compartible relations, which have the same number of attributes and the same attributes domains.
  - Sort-merge technique can be used:
    - Two relations are sorted on the same attributes.
    - A single scan through each relation is sufficient to produce the result.
- The Cartesian Product is quite expensive → avoid using it
  - Substituted by other equivalent operation, if possible.

Query tree

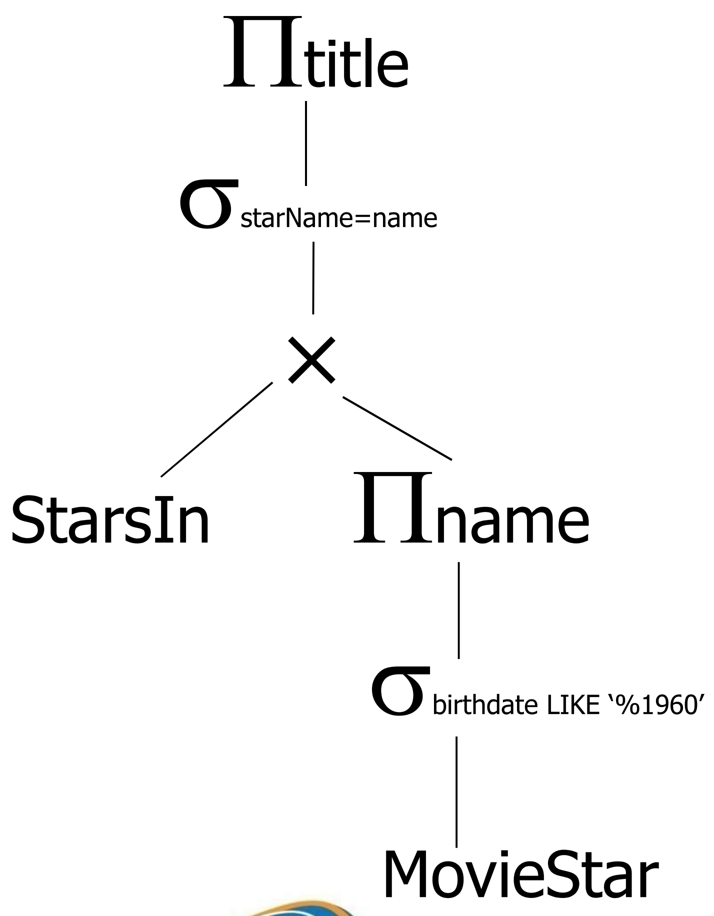


- A query tree is a tree data structure that corresponds to a relational algebra expression.
  - It represents the input relations of the query as leaf nodes of the tree.
  - It represents the relational algebra operations as internal nodes

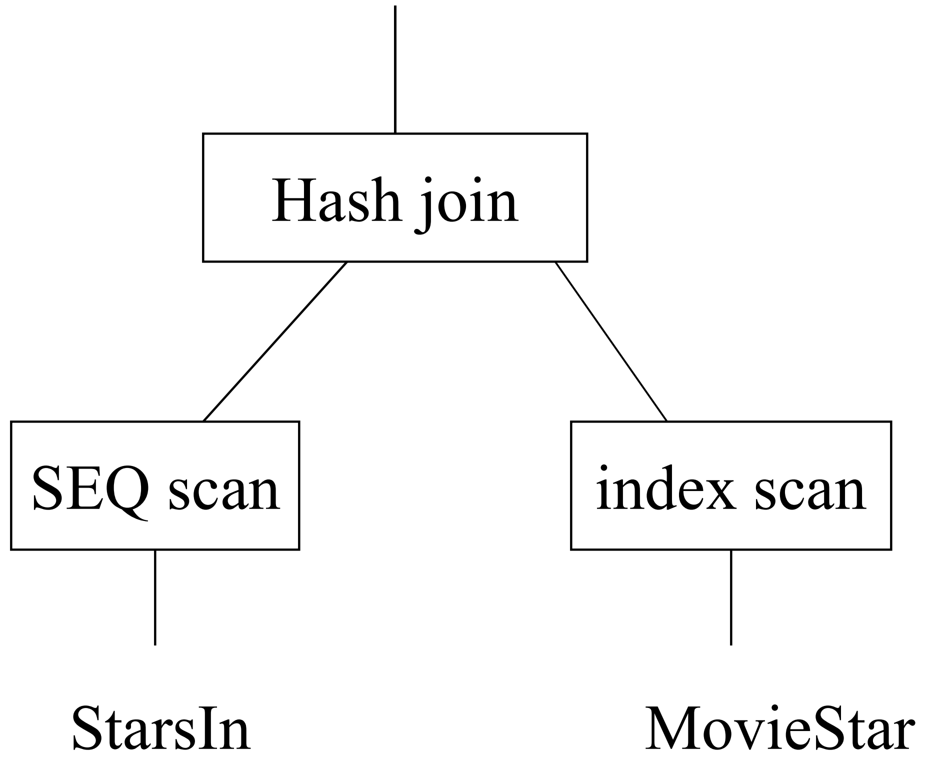
Query plan

- Logical plan (LP): high-level representation of a query, for example query tree.
- Physical plan (PP): low-level representation of a query, by using access methods.
- A logical plan can be implemented by multiple physical plans.

LP AND PP



Π_{title}
$\sigma_{starName=name}$
$\times$
StarsIn
Π_{name}
$\sigma_{birthdate\ LIKE\ '\%1960'}$
MovieStar



Hash join
SEQ scan
index scan
StarsIn
MovieStar

Transformation rules for relational algebra operations

1. $\sigma_{c1 \text{ AND } c2 \text{ AND} \ldots \text{AND } cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(\ldots \sigma_{cn}(R) \ldots))$ broken up the conjunctive selection condition
2. $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ Commutativity of σ
3. $\Pi_{L1}(\Pi_{L2}(\ldots(\Pi_{Ln}(R))\ldots)) \equiv \Pi_{L1}(R)$ Cascade of Π
4. $\Pi_{L1,L2,\ldots,Ln}(\sigma_c(R)) \equiv \sigma_c(\Pi_{L1,L2,\ldots,Ln}(R))$
5. $R_1 \bowtie_c R_2 \equiv R_2 \bowtie_c R_1$ Commutativity of $\bowtie$ and x

   $R_1 x R_2 \equiv R_2 x R_1$
6. $\sigma_c(R_1 \bowtie R_2) \equiv (\sigma_c(R_1)) \bowtie R_2$ Commuting σ with $\bowtie$ (or $\times$)

   $\sigma_c(R_1 \bowtie R_2) \equiv (\sigma_{c1}(R_1)) \bowtie (\sigma_{c2}(R_2))$ if the selection condition c can be written as c1 AND c2, c1 involves all attributes of R1, c2 involves all attributes of R2
7. $\Pi_L(R_1 \bowtie_c R_2) \equiv (\Pi_{A1, A2 \ldots, An}(R_1)) \bowtie_c (\Pi_{B1, B2 \ldots, Bn}(R_2))$ commuting Π with $\bowtie$ (or x) L = {A1, A2, … An, B1, B2, …, Bn} Ai $\in$ R1, Bi $\in$ R2
8. $\cup$ and $\cap$ are commutative, but – is not.
9. Associativity of θ ($\bowtie$, x, $\cup$ and $\cap$): $(R_1 \theta R_2)\theta R_3 \equiv R_1 \theta (R_2 \theta R_3)$
10. $\sigma_c(R_1 \theta R_2) \equiv (\sigma_c(R_1)) \theta (\sigma_c(R_2))$ commuting σ with set operations θ ($\cup$, $\cap$ and -)
11. $\Pi_L(R_1 \cup R_2) \equiv \Pi_L(R_1) \cup \Pi_L(R_2)$
12. $\sigma_c(R_1 x R_2) \equiv R_1 \bowtie_c R_2$ converting a (σ, x) sequence into $\bowtie$
13. DeMorgan's laws: NOT (C1 AND C2) $\equiv$ (NOT C1) OR (NOT C2)

NOT (C1 OR C2) $\equiv$ NOT (C1) AND NOT(C2)

Transformation rules (cont)

14. $(\sigma_P (R_1 - R_2) \equiv \sigma_P (R_1) - R_2$
15. $\Pi_{A1,\dots,An}(\sigma_C(R)) \Leftrightarrow \Pi_{A1,\dots,An}(\sigma_C(\Pi_{A1,\dots,An,Ap} (R)))$

    $\sigma_{c1}(R_1 \bowtie_{c2} R_2) \equiv R_1 \bowtie_{c1 \wedge c2} R_2$

Outline of a Heuristics Algebraic Optimization Algorithm

1. Using rule 1, break up any SELECT opertions with conjunctive conditions into a cascade of SELECT operations. This permit a greater degree of freedom in moving SELECT operations down different branches of the tree.
2. Using rules 2, 4, 6 and 10 concerning the commutativity of SELECT with other operations, move each SELECT operation as far down the query tree as is permitted by the attributes involves in the select condition.
3. Using rules 5 and 9 concerning commutativity and associativity of binary operations, rearrange the leaf nodes of the tree using the following criteria:
   - Executing the SELECT operation as soon as possible, especially the SELECT operation that produces the result with the fewest tuples.
   - Make sure that the ordering of leaf nodes does not cause CARTESIAN PRODUCT operation.

4. Using rule 12, combine a CARTESIAN PRODUCT operation with a subsequent SELECT operation in the tree into a JOIN operation, if possible.
5. Using rules 3, 4, 7, 11, break down and move lists of projection attributes down the tree as far as possible by creating new PROJECTION operations as needed.
6. Identify subtrees that represent groups of operations that can be executed by a single algorithm.

Using selectivity and cost estimates in query optimization

- The query optimizer should also estimate and compare the costs of executing a query using different execution strategies and should choose the strategy with the lowest cost estimate.
- Cost components for query execution:
  1. Access cost to secondary storage: searching, reading, writing, …
  2. Storage cost: cost of storing any intermediate files.
  3. Computation cost: cost of performing in-memory operations on the data buffer.
  4. Memory usage cost: cost pertaining to the number of memory buffers needed.
  5. Communication cost: cost of shipping query and its results.

Using selectivity and cost estimates in query optimization

- Catalog information used in Cost functions.
  - Number of records r.
  - Record size R.
  - Number of blocks b.
  - Blocking factor bfr.
  - Index information.
  - …

End.