Query Processing

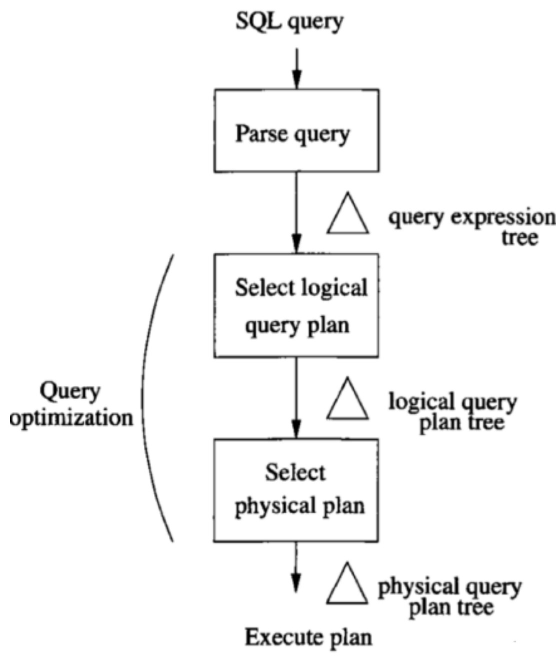
Dr Eugenia Ternovska
Associate Professor

Simon Fraser University

Query Processing

Query → Parse Tree → Logical Query Plan → Physical Query Plan → Query Execution

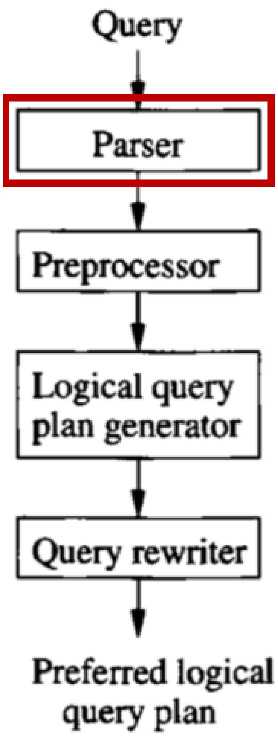
- Parse Tree
- Logical Query Plan
  - Logical Optimization: Improvement through algebraic laws
- Physical Query Plan
  - Cost Estimation
  - Algorithm choice
  - Pipelining versus materialization



Parsing

- Take SQL and convert it to a parse tree
- Parse tree
  - Atoms
  - No Children
  - Keywords, names, constants, parentheses, operators
- Syntactic categories
  - Have Children

    `<Query>, <Condition>`



Parsing Process Rules (1)

```
<Query> ::= SELECT <SelList> FROM
               <FromList> WHERE <Condition>
```

Symbol ::= means “can be expressed as”

```
 • <SelList>
Either a single attribute or an attribute, a comma,
and any list of one or more attributes.
• <FromList>
Any comma-separated list of relations
• <Condition>
Represents SQL conditions
```

Parsing Process Rules (2)

```
<Query> ::= SELECT <SelList> FROM
                <FromList> WHERE <Condition>
```

- <Attribute>
  Any string of characters that identifies an attribute of the current database schema
- <Relation>
  Can be replaced by any string of characters that makes sense as a relation in the current schema
- <Pattern>
  Can be replaced by any quoted string that is a legal SQL pattern

Sample Parsing Process Rules

```
<SelList> ::= <Attribute> , <SelList>
<SelList> ::= <Attribute>

<FromList> ::= <Relation> , <FromList>
<FromList> ::= <Relation>

<Condition> ::= <Condition> AND <Condition>
<Condition> ::= <Attribute> IN ( <Query> )
<Condition> ::= <Attribute> = <Attribute>
<Condition> ::= <Attribute> LIKE <Pattern>
```

Example

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

Query: find the titles of movies that have at least one star born in 1960

(A)

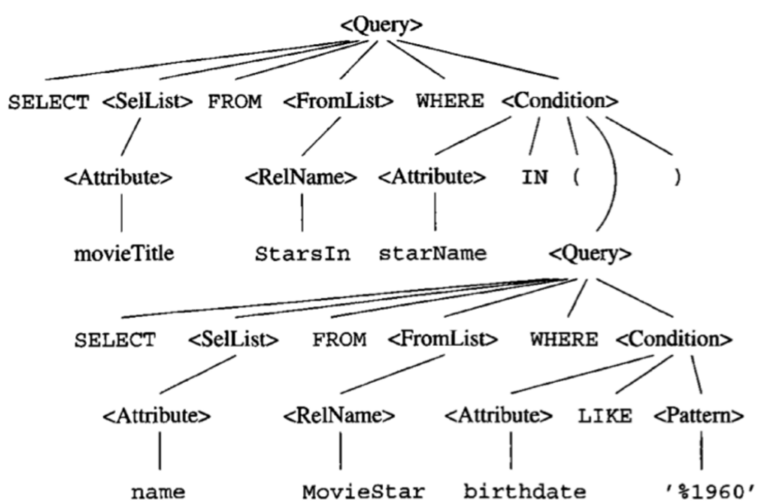
```
SELECT movieTitle
FROM StarsIn
WHERE starName IN (
SELECT name
FROM MovieStar
WHERE birthdate
    LIKE '%1960'
);
```

(B)

```
SELECT movieTitle
FROM StarsIn, MovieStar
WHERE starName = name AND
birthdate LIKE '%1960';
```

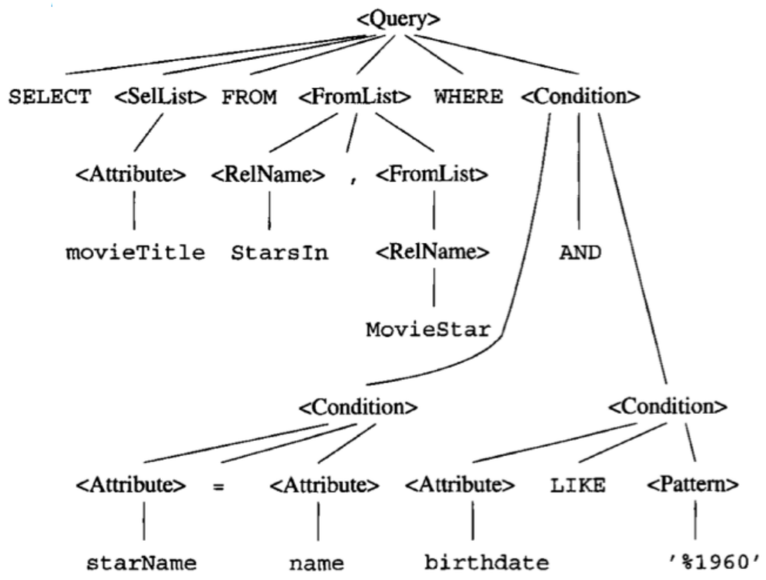
Example (A)

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



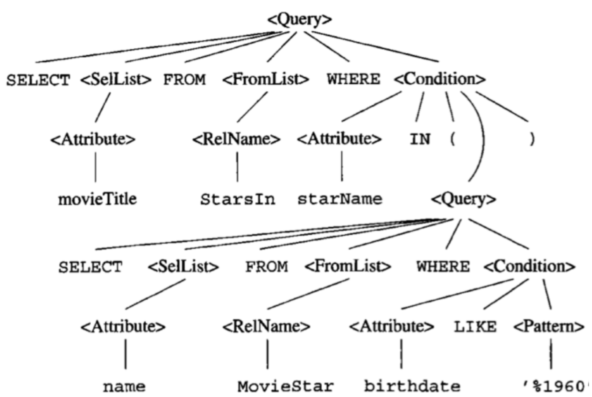
Example (B)

```
SELECT movieTitle
FROM
StarsIn, MovieStar
WHERE
starName = name AND
birthdate
LIKE '%1960';
```

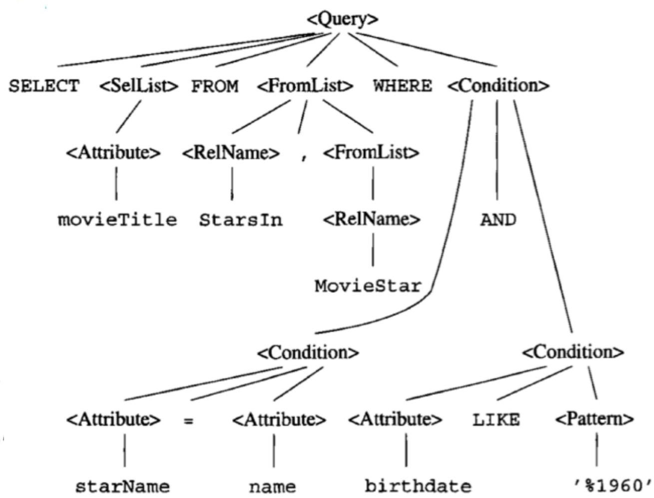


Example

(A)

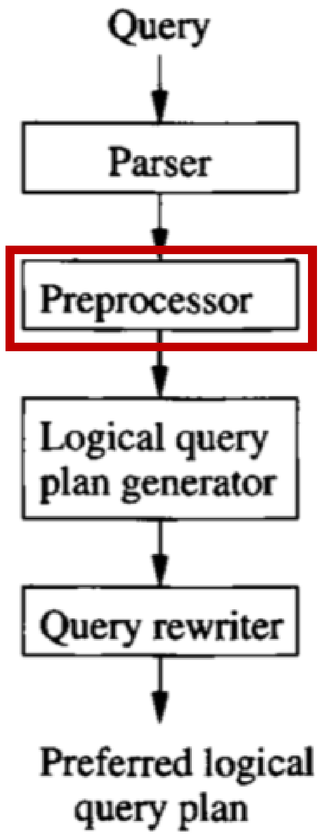


(B)



Preprocessor

- Obtain Parse Tree for Views
  • If a relation is a virtual view replace it by a parse tree for view (a query)
- Perform Semantic Checking
  • Check relation uses: every relation mentioned is a relation or view in the current schema
  • Check and resolve attribute uses: every attribute used is an attribute of some relation in the current scope
  • Check types: type appropriate to their uses



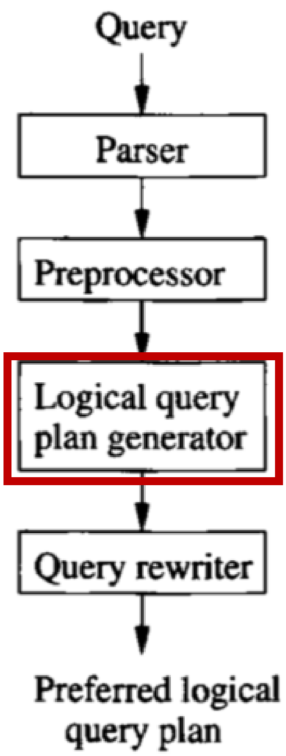
Conversion to Relational Algebra

If a **<Query>** with a **<Condition>** has no subqueries, We may replace it by a RA expression consisting, from bottom to top.

- The product of all the relations mentioned in the **<FromList>**, which is the argument of
- A selection σ_C, where C is the **<Condition>** expression in the construct being replaced, which in turn is the argument of
- A projection π_L, where L is the list of attributes in the **<SelList>**

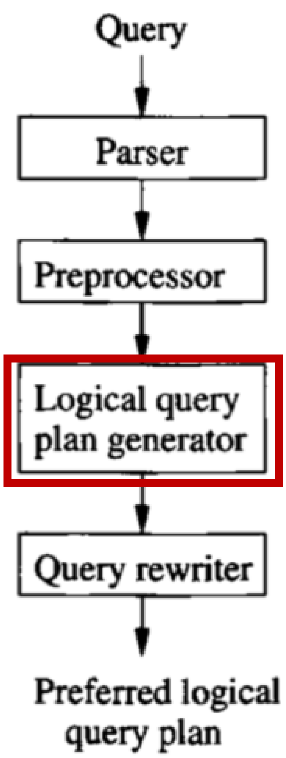
Logical Query

- Conversion to Relational Algebra using algebraic Laws for Improving Query Plans
- Removing Subqueries From Conditions
- Improving the Logical Query Plan (Rewriting)



Some Trivial Laws

- Any selection on an empty relation is empty
- If C is an always-true condition then $\sigma_C(R) = R$
(e.g. $x > 10$ OR $x =< 10$ on a relation that forbids x = NULL)
- If R is empty, then R $\cup$ S = S
- Order of operator and argument presentation
In Relational Algebra: X, $\cup$, $\cap$, and $\bowtie$ are commutative and associative.



Laws Involving Selection

Selections reduce the size of relations

- Push down laws:
  move the selections down the tree as far as they will go without changing what the expression does (Push down selection: Major tool for optimizer)
- Split laws:
  when the condition of a selection is complex:

Push Laws (Selection)

- For a union the selection must be pushed to both arguments
  $\sigma_C\ (R \cup S) = \sigma_C\ (R) \cup \sigma_C\ (S)$
- For a difference the selection must be pushed to the first argument and optionally may be pushed to the second
  $\sigma_C\ (R - S) = \sigma_C\ (R) - S$
  $\sigma_C\ (R - S) = \sigma_C\ (R) - \sigma_C\ (S)$
- For the other operators it is only required that the selection be pushed to one argument
  $\sigma_C\ (R \cap S) = \sigma_C\ (R) \cap S$

Split Laws (Selection)

$\sigma_{C_1 \text{ AND } C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$

$\sigma_{C_1 \text{ OR } C_2}(R) = (\sigma_{C_1}(R)) \cup_S (\sigma_{C_2}(R))$
We use U_S to indicate that it is only a set union

$\sigma_{C_1}(\sigma_{C_2}(R)) = \sigma_{C_2}(\sigma_{C_1}(R))$

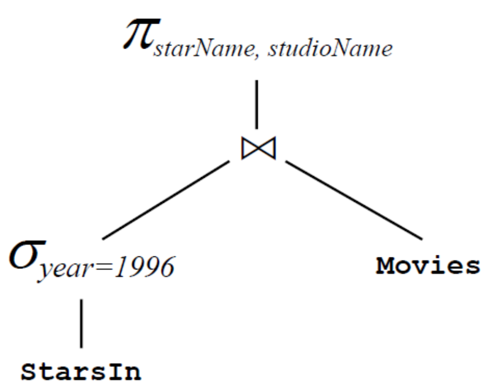
Pushing Selections

```
CREATE VIEW MoviesOf1996
AS SELECT *
FROM Movies
WHERE year = 1996;

SELECT starName, studioName
FROM MoviesOf1996
    NATURAL JOIN StarsIn
```

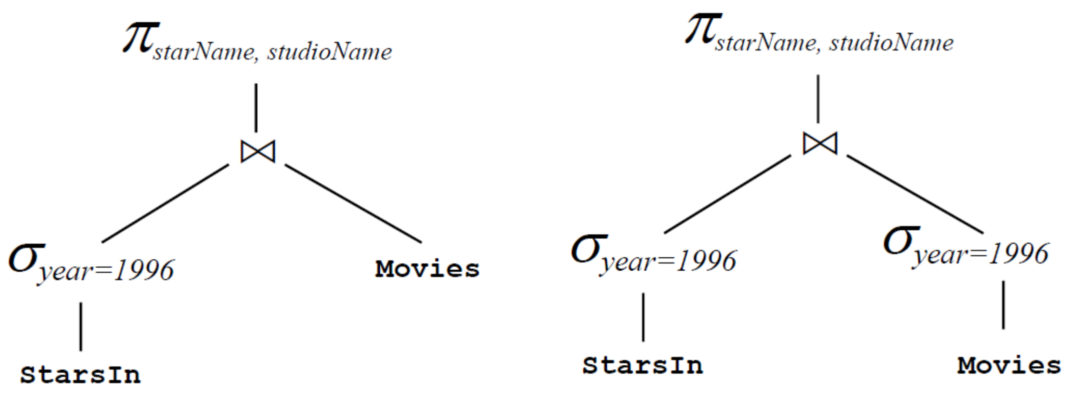
Push Law:

$\sigma_{year=1996}(Movies \bowtie StarsIn)$
$= (\sigma_{year=1996}(Movies)) \bowtie StarsIn$



(a) Logical query plan constructed from definition of a query and view

Pushing Selections



(a) Logical query plan constructed from definition of a query and view

(b) Improving the query plan by moving selections up & down the tree

Laws Involving Projection

Introducing a new projection somewhere below an existing projection

- We may introduce a projection anywhere in an expression tree, as long as it eliminates only attributes that are neither used by an operator above nor are in the result of the entire expression

  $\pi_L\ (R \bowtie S) = \pi_L\ (\pi_M\ (R) \bowtie \pi_N(S))$
  $\pi_L\ (R \bowtie_C S) = \pi_L\ (\pi_M\ (R) \bowtie_C \pi_N(S))$
  $\pi_L\ (R \times S) = \pi_L\ (\pi_M\ (R) \times \pi_N(S))$

  M and N are the join attributes and the input attributes of L that are found among the attributes of R and S, respectively
- Less useful than pushing selections

Laws About Joins and Products

- Commutative & associative laws
- Some additional laws
  $R \bowtie_C S = \sigma_C(R \times S)$
  $R \bowtie S = \pi_L\ (\sigma_C(R \times S))$
  C is the condition that equates each pair of attributes from R and S with the same name, and L is a list that includes one attribute from each equated pair and all the other attributes of R and S

Additional Laws

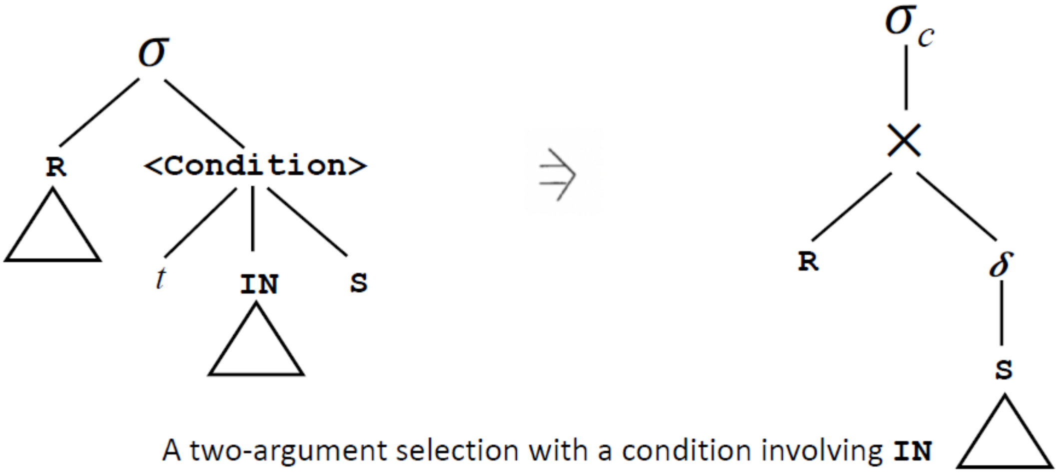
- Laws Involving Duplicate Elimination (δ)
  • Can be pushed through many, but not all operators
  • We can sometimes move it to where it can be eliminated completely
- Laws Involving Grouping and Aggregation (γ)
  • Depends on the details of the aggregate: Not many general rules
  • We may project useless attributes prior to applying

Improving the Logical Query Plan

- Selections can be pushed down the expression tree as far as they can go
- Projections can be pushed down the tree, or new projections can be added
- Duplicate eliminations can sometimes be removed, or moved to a more convenient position in the tree
- Selections can be combined with a product below to turn the pair of operations into an equijoin

Removing Subqueries from Conditions

- Two-argument selection
  - A node labeled σ, with no parameter
  - Left child: The relation R upon which the selection is being performed
  - Right child: An expression for the condition applied to each tuple of R

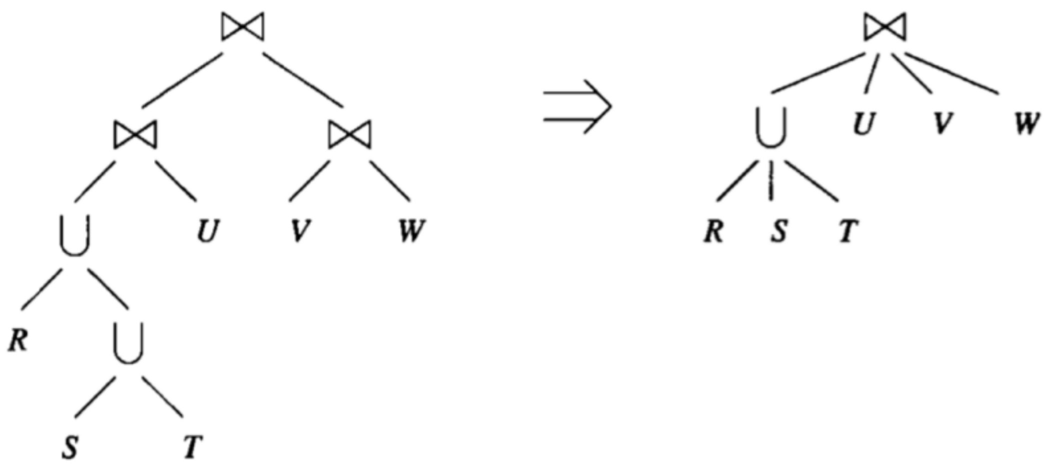


A two-argument selection with a condition involving **IN**

Final Steps

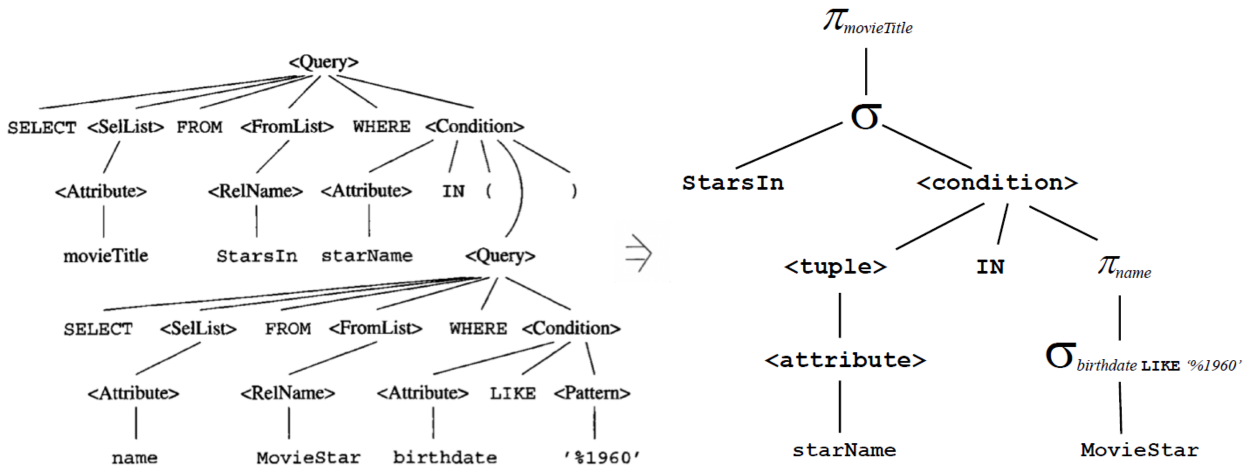
- Grouping Associative/Commutative Operators
  - Replace the natural joins with theta-joins that equate the attributes of the same name
  - Must add a projection to eliminate duplicate copies of attributes involved in a natural join that has become a theta-join
  - The theta-join conditions must be associative

Final Steps

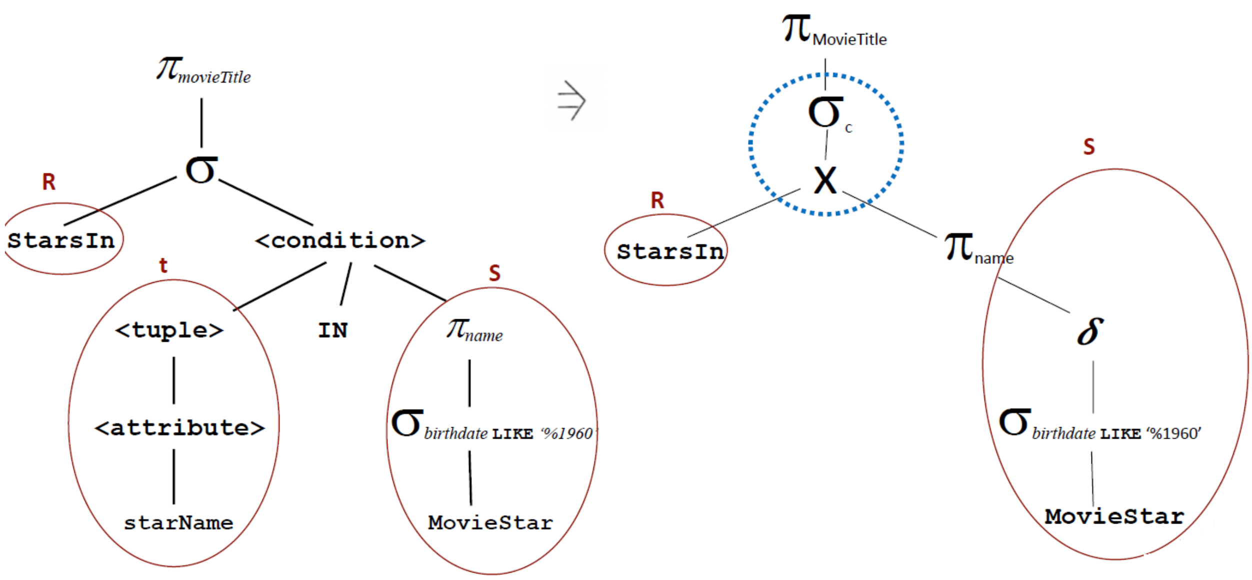


Final step in producing the logical query plan:
group the associative and commutative operators

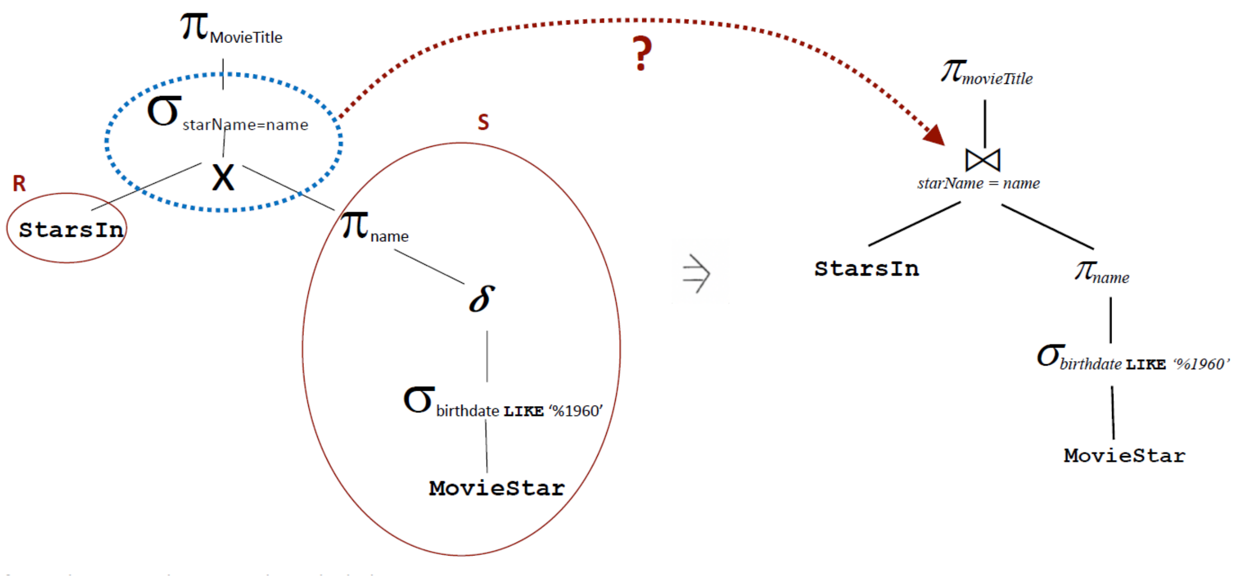
Example (1)



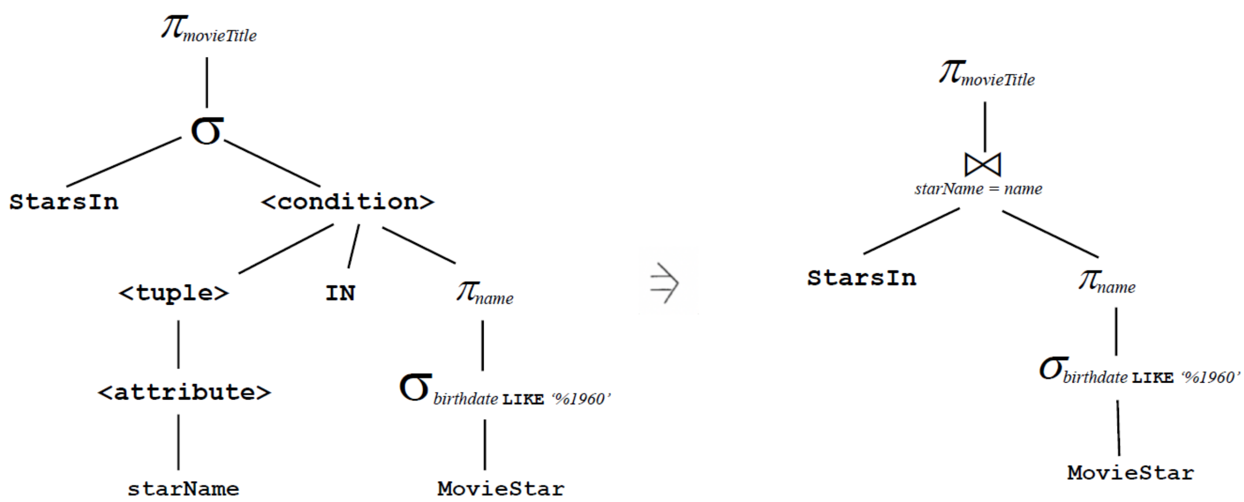
Example (2)



Example (3)



Example (4)



From Parse Trees to Logical Query Plans

- Logical Query Plan Generator
  - Conversion to Relational Algebra
  - Removing Subqueries From Conditions
  - Improving the Logical Query Plan
  - Grouping Associative & Commutative Operators

Logical Plan to Physical Plan

- Estimating the Cost of Operations
  - Cost-based enumeration:
  - An order and grouping for associative-and-commutative operations (e.g. joins, unions, intersections)
  - An algorithm for each operator in the logical plan (e.g. deciding whether a nested-loop join or a hash-join should be used)
  - Additional operators (e.g. scanning, sorting) that are needed for the physical plan but that were not present explicitly in the logical plan
  - The way in which arguments are passed from one operator to the next (e.g. by storing the intermediate result on disk or by using iterators and passing an argument one tuple or one main-memory buffer at a time)
- Estimating Sizes of Intermediate Relations

Size Estimation (1)

Given:
B(R): The number of blocks needed to hold relation R
T(R): Is the number of tuples of relation R
V(R, a): Is the value count for attribute a of relation R, that is, the number of distinct values relation R has in attribute a

- Estimating size of a selection:
  • Let $S = \sigma_{A=C}(R)$, where A is an attribute of R and C is a constant.
  An estimate $T(S) = T(R)/V(R,A)$
- Estimating size of a natural join:
  $T(R \bowtie S) = T(R)T(S)/\max(V(R, Y), V(S, Y))$

Size Estimation (2)

- Union
  • As large as the sum of the sizes or as small as the larger of the two arguments
  • One approach: Use an average
- Intersection
  • As few as 0 tuples or as many as the smaller of the two arguments
  • One approach: Take the average of the extremes, which is half the smaller
- Difference
  • When we compute R—S, the result can have between T(R) and T(R)—T(S) tuples
  • One approach: Average as an estimate: T(R)—T(S)/2

Example

- R(a,b) has T(R)=10,000
- S=$\sigma_{\text{a=10 OR b<20}}$ (R)
- Let V(R,a)=50
- The number of tuples that satisfy b<20 is 1/3 of the total tuples
- What is the estimate for the size of S?

Enumerating Physical Plans

- Top-down
  - Work down the tree of the logical query plan from the root
- Bottom-up
  - Compute the costs of all possible ways to compute that subexpression
  - Combine them in all possible ways

Acknowledgements

[1] Database Systems: The Complete Book, 2nd Edition Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer WidomPrentice Hall, 2009

[2] Database System Concepts, Seventh EditionAvi Silberschatz, Henry F. Korth, S. SudarshanMcGraw-Hill, March 2019www.db-book.com

Additional references and resources used in preparation of this course are listed on the course webpage or mentioned in slides.