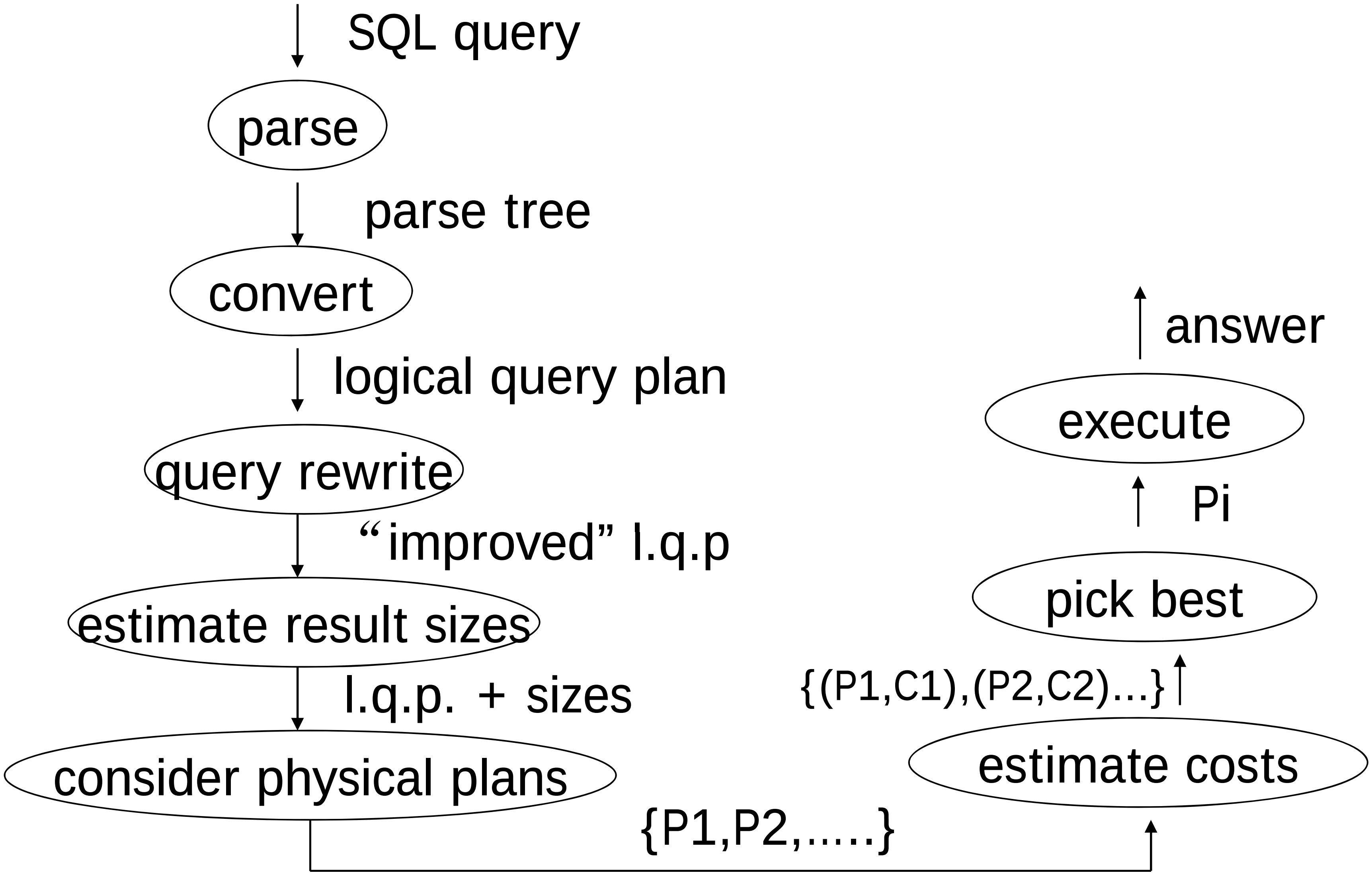
Query Processing and Advanced Queries

Query Processing (2)

Review: Query Processing



SQL query

parse

parse tree

convert

logical query plan

query rewrite

“improved” l.q.p

estimate result sizes

l.q.p. + sizes

consider physical plans

{P1,P2,.....}

estimate costs

{(P1,C1),(P2,C2)...}

pick best

Pi

execute

answer

Parsing

Grammar for SQL

- The following grammar describes a simple subset of SQL.
- Queries

  <Query>::= SELECT <SelList> FROM <FromList>
  WHERE <Condition> ;
- Selection lists

  <SelList>::= <Attribute>, <SelList>
  <SelList>::= <Attribute>
- From lists

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

Parsing

Grammar for SQL

- Conditions

  <Condition>::= <Condition> AND <Condition>

  <Condition>::= <Attribute> IN (<Query>)

  <Condition>::= <Attribute> = <Attribute>

  <Condition>::= <Attribute> LIKE <Pattern>

- Syntactic categories Relation and Attribute are not defined by grammar rules, but by the database schema.
- Syntactic category Pattern defined as some regular expression.

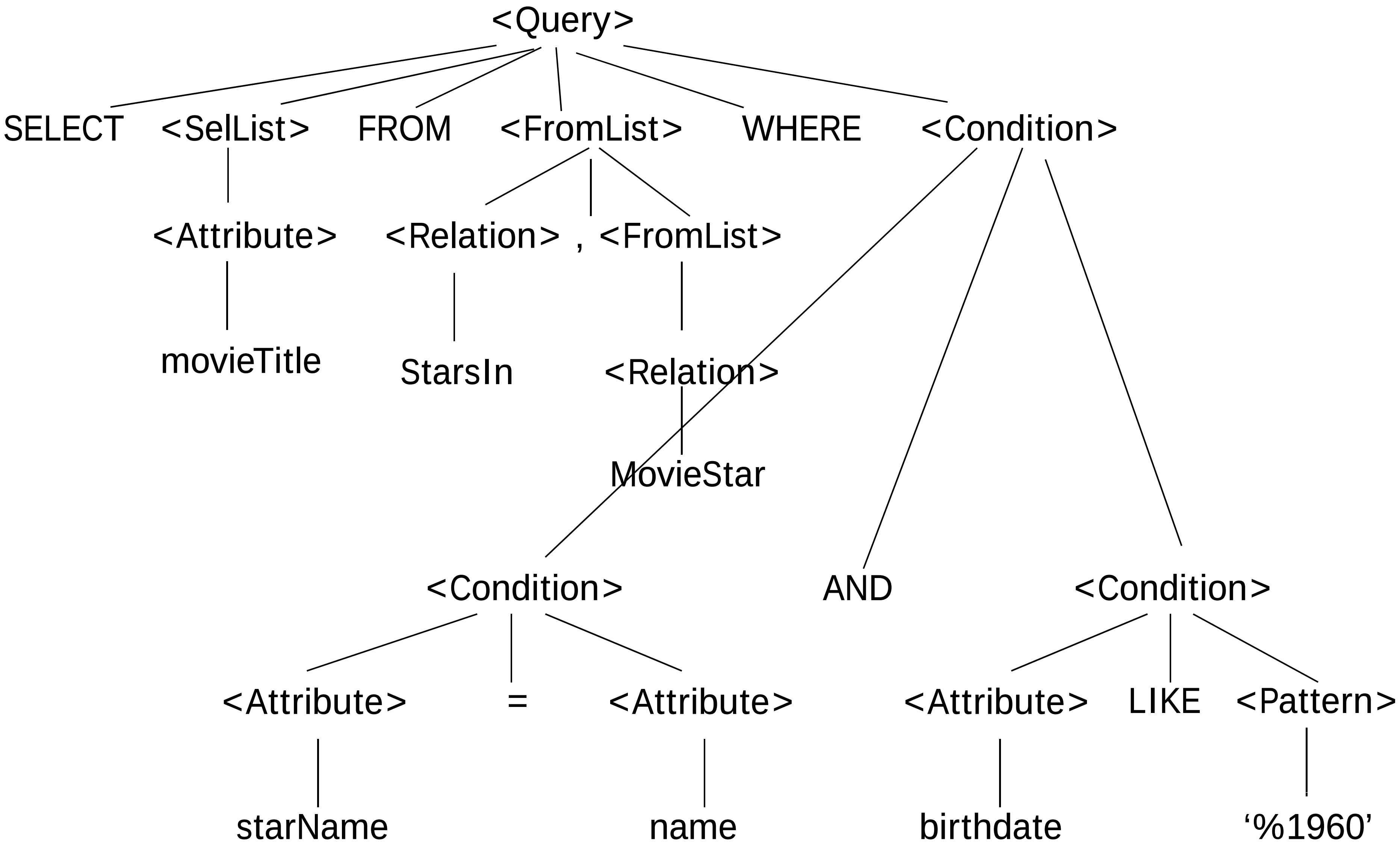
Example: A SQL Query

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

Goal: find the movies with stars born in 1960

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

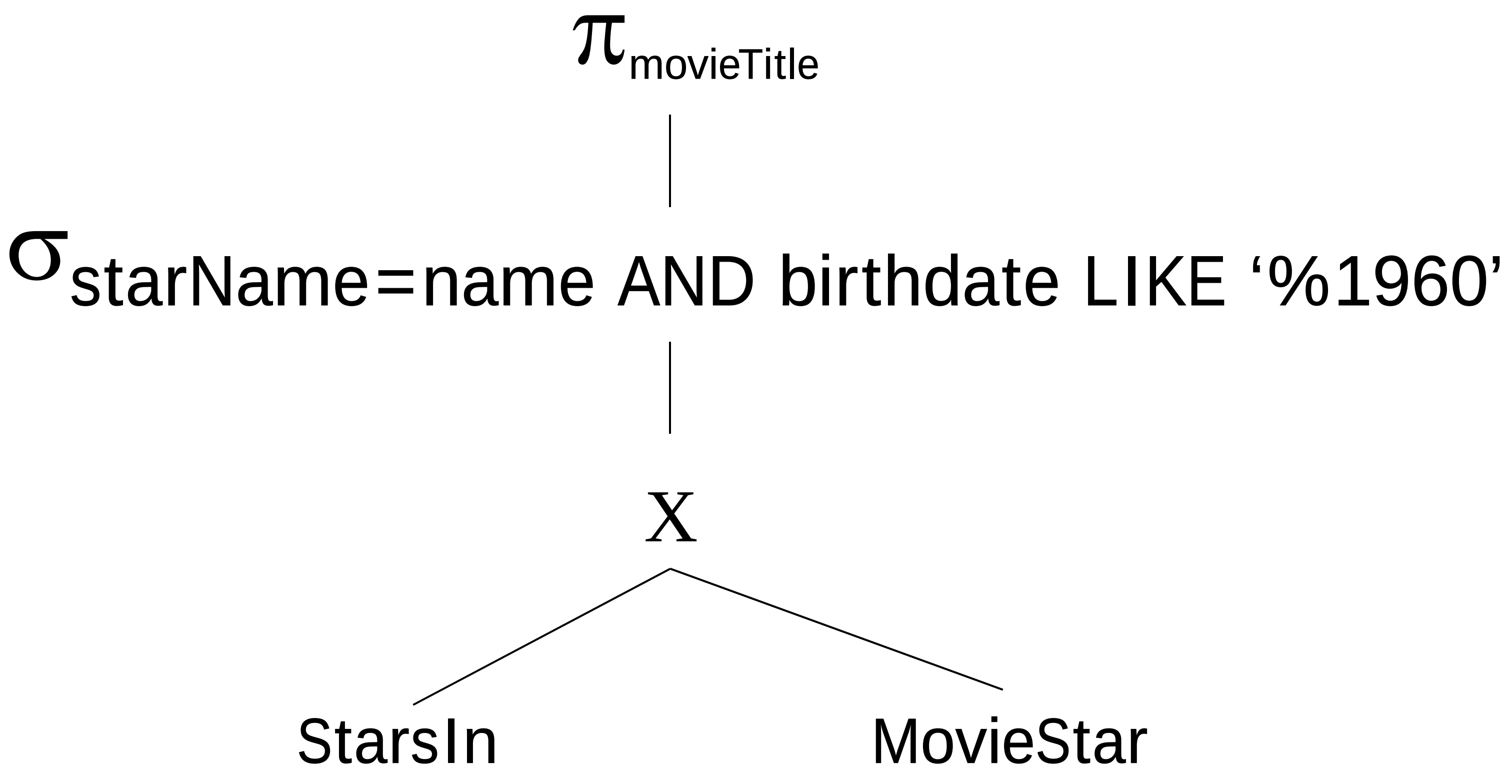
A Parse Tree


<Query>
SELECT
<SelList>
FROM
<FromList>
WHERE
<Condition>
<Attribute>
<Relation>
,
<FromList>
movieTitle
StarsIn
<Relation>
MovieStar
<Condition>
AND
<Condition>
<Attribute>
=
<Attribute>
<Attribute>
LIKE
<Pattern>
starName
name
birthdate
‘%1960’

Conversion to Logical Query Plan

- How to convert a parse tree into a logical query plan, i.e. a relational algebra expression?
- Queries with conditions without subqueries are easy:
  - Form **Cartesian product** of all relations in <FromList>.
  - Apply a **selection σ_c** where C is given by <Condition>.
  - Finally apply a **projection π_L** where L is the list of attributes in <SelList>.
- Queries involving subqueries are more difficult.
  - Remove subqueries from conditions and represent them by a two-argument selection in the logical query plan.

An Algebraic Expression Tree



$\pi_{\text{movieTitle}}$

$\sigma_{\text{starName=name AND birthdate LIKE ‘\%1960’}}$

X

StarsIn

MovieStar

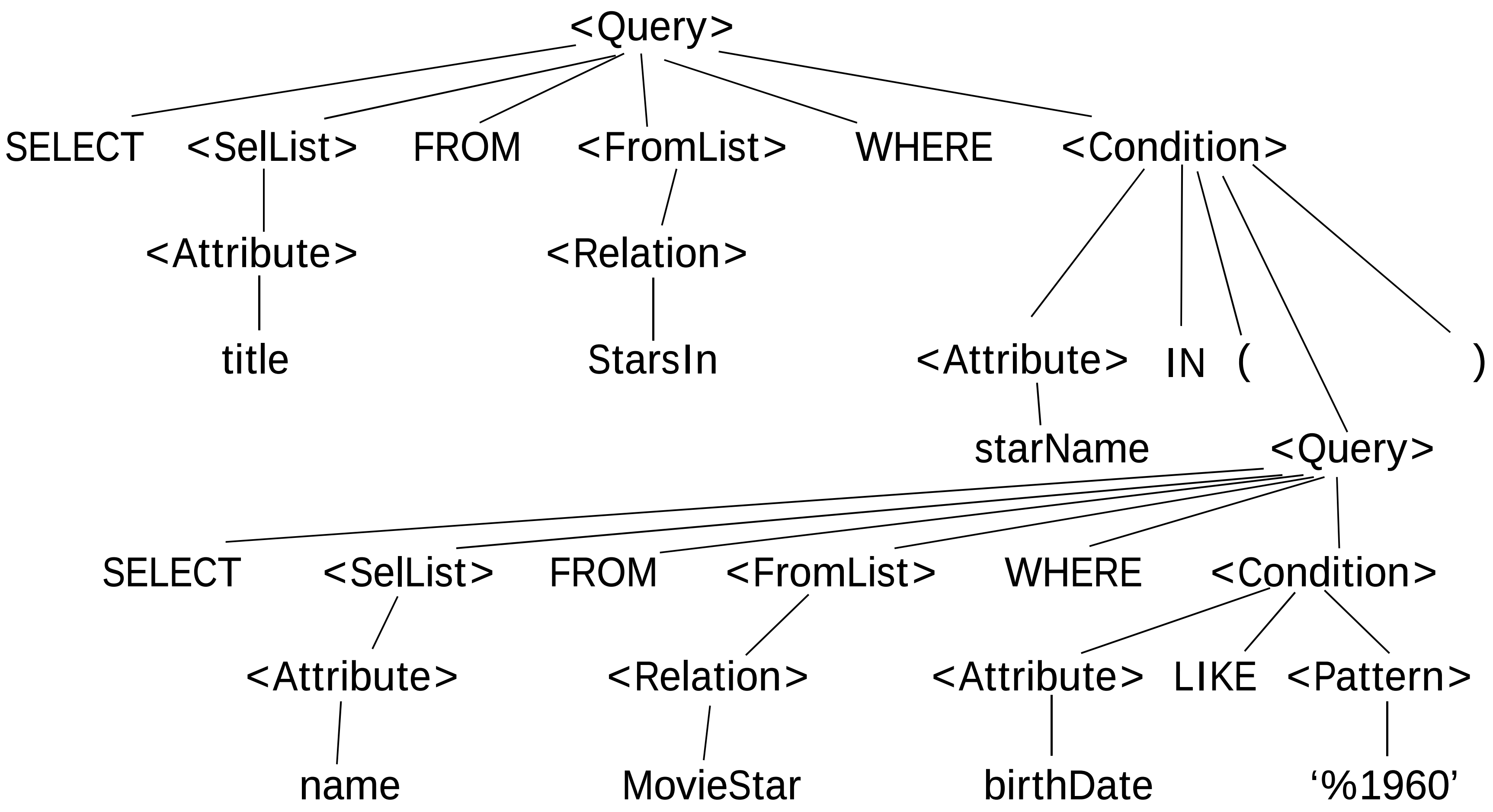
Another SQL Query

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

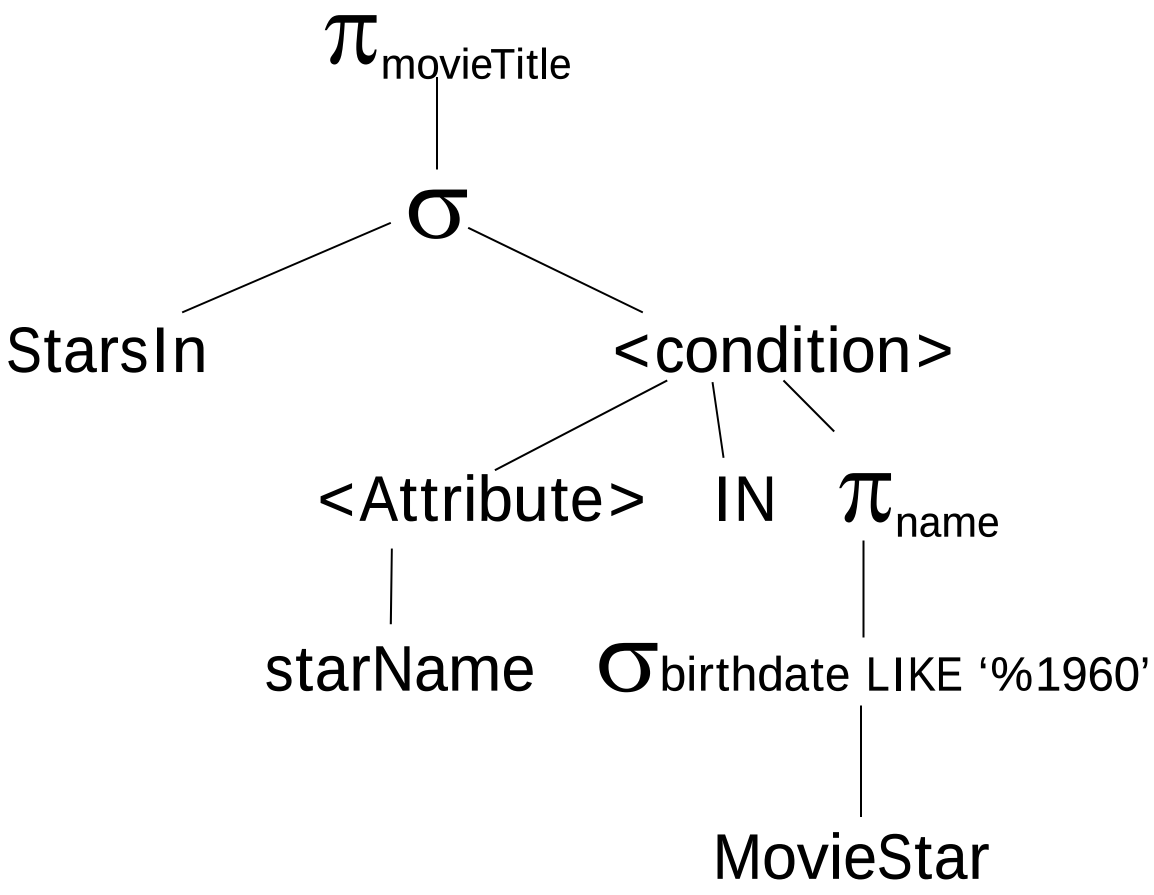
Goal: find the movies with stars born in 1960

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

Another Parse Tree



Another Algebraic Expression Tree



Algebraic Laws for Query Plans

Introduction

- Algebraic laws allow us to transform a Relational Algebra (RA) expression into an equivalent one.
- Two RA expressions are *equivalent* if, for all database instances, they produce the same answer.
- The resulting expression may have a more efficient physical query plan.
- Algebraic laws are used in the query rewrite phase.

Algebraic Laws for Query Plans

Introduction

- *Commutative law*:

  Order of arguments does not matter.

  $$x + y = y + x$$

- *Associative law*:

  May group two uses of the operator either from the left or the right.

  $$(x + y) + z = x + (y + z)$$

- Operators that are commutative and associative can be grouped and ordered arbitrarily.

Algebraic Laws for Query Plans

Natural Join, Cartesian Product and Union

$R \bowtie S = S \bowtie R$

$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

R x S = S x R

(R x S) x T = R x (S x T)

R U S = S U R

R U (S U T) = (R U S) U T

Algebraic Laws for Query Plans

Selection

$$\sigma_{p1 \wedge p2}(R) = \sigma_{p1} \; [\, \sigma_{p2}\, (R)]$$

$$\sigma_{p1 \vee p2}(R) = [\, \sigma_{p1}\, (R)] \cup [\, \sigma_{p2}\, (R)]$$

$$\sigma_{p1} \; [\, \sigma_{p2}\, (R)] = \sigma_{p2}\, [\, \sigma_{p1}\, (R)]$$

- Simple conditions p1 or p2 may be pushed down further than the complex condition.

Algebraic Laws for Query Plans

Bag Union

- What about the union of relations with duplicates (bags)?

  R = {a,a,b,b,b,c}
  S = {b,b,c,c,d}
  R U S = ?

- Number of occurrences either SUM or MAX of occurrences in the imput relations.

  SUM: R U S = {a,a,b,b,b,b,b,c,c,c,d}
  MAX: R U S = {a,a,b,b,b,c,c,d}

Algebraic Laws for Query Plans

Selection

- $\sigma_{p1 \vee p2}(R) = \sigma_{p1}(R) \cup \sigma_{p2}(R)$
- MAX implementation of union makes rule work.
- R={a,a,b,b,b,c}

  p1 satisfied by a,b, p2 satisfied by b,c

  $\sigma_{p1 \vee p2}(R) = \{a,a,b,b,b,c\}$

  $\sigma_{p1}(R) = \{a,a,b,b,b\}$

  $\sigma_{p2}(R) = \{b,b,b,c\}$

  $\sigma_{p1}(R) \cup \sigma_{p2}(R) = \{a,a,b,b,b,c\}$

Algebraic Laws for Query Plans

Selection

- $\sigma_{p1 \vee p2}(R) = \sigma_{p1}(R) \cup \sigma_{p2}(R)$
- SUM implementation of union makes more sense.

Senators (……) Reps (……)

$T1 = \pi_{yr,state}$ Senators, $T2 = \pi_{yr,state}$ Reps

T1	Yr	State
	97	CA
	99	CA
	98	AZ

T2	Yr	State
	99	CA
	99	CA
	98	CA

Union?

- **Use** SUM implementation, but then some laws do not hold.

Algebraic Laws for Query Plans

Selection and Set Operations

$\sigma_p(R \text{ U } S) = \sigma_p(R) \text{ U } \sigma_p(S)$

$\sigma_p(R - S) = \sigma_p(R) - S = \sigma_p(R) - \sigma_p(S)$

Algebraic Laws for Query Plans

Selection and Join

- p: predicate with only R attributes
  q: predicate with only S attributes
  m: predicate with attributes from R and S

- $\sigma_p (R \bowtie S) = [\sigma_p (R)] \bowtie S$

  $\sigma_q (R \bowtie S) = R \bowtie [\sigma_q (S)]$

Algebraic Laws for Query Plans

Selection and Join

$$\sigma_{p \wedge q} (R \bowtie S) = [\sigma_p (R)] \bowtie [\sigma_q (S)]$$

$$\sigma_{p \wedge q \wedge m} (R \bowtie S) =$$

$$\sigma_m \left[(\sigma_p R) \bowtie (\sigma_q S)\right]$$

$$\sigma_{p v q} (R \bowtie S) =$$

$$\left[(\sigma_p R) \bowtie S\right] \cup \left[R \bowtie (\sigma_q S)\right]$$

Algebraic Laws for Query Plans

Projection

- X: set of attributes

  Y: set of attributes

  XY: X U Y

- $\pi_{xy}(R) = \xcancel{\pi_x[\pi_y}(R)]$

- May introduce projection anywhere in an expression tree as long as it eliminates no attributes needed by an operator above and no attributes that are in result

Algebraic Laws for Query Plans

Projection and Selection

- X: subset of R attributes
  Z: attributes in predicate P (subset of R attributes)

- $\pi_x (\sigma_p R) = \pi_x \{\sigma_p [\overset{\pi_{xz}}{\cancel{\pi_x}} (R)]\}$

- Need to keep attributes for the selection and for the result

Algebraic Laws for Query Plans

Projection and Selection

- X: subset of R attributes
- Y: subset of S attributes
- Z: intersection of R,S attributes

- $\pi_{xy}(R \bowtie S) =$

$$\pi_{xy}\{[\pi_{xz}(R)] \bowtie [\pi_{yz}(S)]\}$$

Algebraic Laws for Query Plans

Projection, Selection and Join

$$\pi_{xy}\{\sigma_p\ (R \bowtie S)\} =$$

$$\pi_{xy}\{\sigma_p\ [\pi_{xz'}\ (R) \bowtie \pi_{yz'}\ (S)]\}$$

$$z' = z \cup \{\text{attributes used in } P\}$$

What Are Good Transformation?

- No transformation is always good
- Usually good: early selections/projections