

CS525: Advanced Database Organization

Notes 6: Query Processing Part III: Convert Parse Tree into initial L.Q.P

Yousef M. Elmehdwi

Department of Computer Science

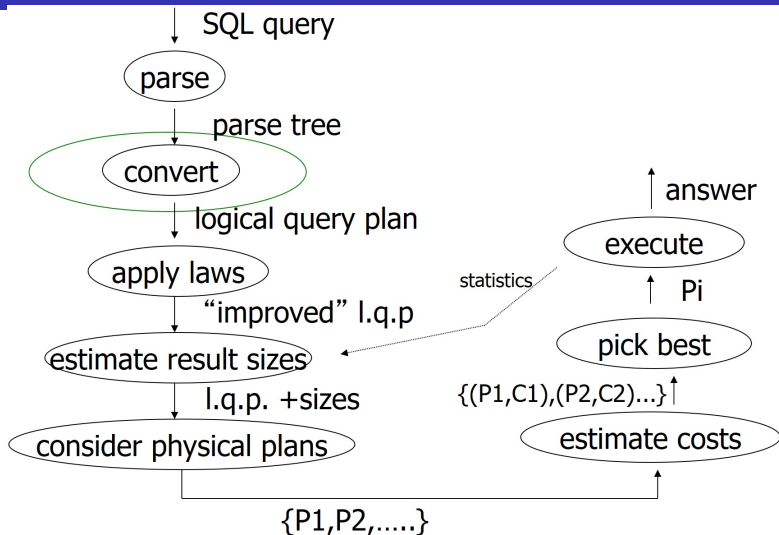
Illinois Institute of Technology

yelmehdwi@iit.edu

November 15st 2023

Slides: adapted from a course taught by [Hector Garcia-Molina](#), Stanford, [Ellen Munthe-Kaas](#), Universitetet Oslo & [T. Murali](#), Virginia Polytechnic Institute

Where we are?



- *Note: when we have executed the query, it might be wise to give statistics back to LQP-rewrite components or components estimating size to perform later operations like join in a cost-efficient order*

Today

- Convert the **parse query tree** to an **initial query plan**, which is usually an algebraic representation of the query (relational algebra expression)

Relation Schema, Relation Instance, Tuple

- Relation

- Unordered set that contain the relationship of attributes that represent entities.

- Tuple

- Set of attribute values (also known as its domain) in the relation/row of values of the relation (table).
 - Values are normally atomic/scalar.
 - The special value **NULL** is a member of every domain.

- Relation Schema

- A set of **attribute name-datatype** pairs
- Schema for a relation defined the names of the attributes and the domain for the attributes

- *Much of query optimization relies on the underlying concept that the high level properties of relational algebra are preserved across equivalent expressions. Two relational algebra expressions are equivalent if they generate the same set of tuples.*
- Create an internal representation
 - Should be useful for analysis
 - Should be useful optimization
- Internal representation
 - Relational algebra
- *Creating an effective internal representation for query optimization in a database management system is crucial for efficient query processing. This internal representation should not only be useful for analysis but also support optimization.*

Relational Algebra Recap

- A set of fundamental operations for retrieving and manipulating tuples in a relation.
- It serves as a mathematical language for working with sets, based on set algebra principles.
- Operands are relations or variables representing relations.
- Operators are designed for common database tasks.
- It acts as an abstract query language for relations.
- Operators take relations as inputs and produce new relations as outputs.
- By chaining operators, complex operations and queries can be created.

Set- vs. Bag semantics

- Set and Bag semantics version
 - *Set*: a collection of elements that do not contain duplicate elements
 - *Bag*: a collection of elements that can contain duplicate elements
- Relational Algebra has two semantics:
 - **Set semantics**: Relations are **Sets** of tuples, which by definition do not contain “duplicate” entries
 - **Bag semantics**: Relations are **Multi-Sets** where each element (tuple) can appear more than once
 - duplicates are not eliminated
- SQL uses **bag semantics**
- If we use **bag semantics**, we may have to redefine the meaning of each relation algebra operator

Set- vs. Bag semantics

Set

Name	Purchase
Peter	Guitar
Joe	Drum
Alice	Bass

Bag

Name	Purchase
Peter	Guitar
Peter	Guitar
Joe	Drum
Alice	Bass
Alice	Bass

Why Bags?

- SQL, the most important query language for relational databases is actually a **bag** language.
 - SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.
- Some operations, like **projection** or **union**, are much more efficient on **bags** than sets
 - **Selection** applies to each tuple, so its effect on bags is like its effect on sets.
 - **Projection** also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
 - **Products** and **joins** are done on each pair of tuples, so duplicates in bags have no effect on how we operate

Bag semantics notation

- We use t^m to denote tuple t appears with multiplicity m

Operators

- Selection
- Renaming
- Projection
- Joins
 - Theta, natural, cross-product, outer, anti
- Aggregation
- Duplicate removal
- Set operations

Selection Operation (σ)

- Choose a subset of the tuples from a relation that satisfies a selection predicate.
 - Predicate acts as a filter to retain only tuples that fulfill its qualifying requirement.
 - Can combine multiple predicates using conjunctions / disjunctions.
- **Syntax:** $\sigma_c(R)$
 - R is input
 - c is a condition (called the selection predicate)
- **Semantics:**
 - Return all tuples that match condition c
 - **Set:** $\{ t \mid t \in R \text{ AND } t \text{ fulfills } c \}$
 - **Bag:** $\{ t^n \mid t^n \in R \text{ AND } t \text{ fulfills } c \}$
 - process each tuple independently; a tuple may appear in the resulting relation multiple times.

Select: Example

- $\sigma_{a>5}(R)$

R

a	b
1	13
3	12
6	14

Result

a	b
6	14

Projection operation (π)

- Generate a relation with tuples that contains only the specified attributes.
 - Can rearrange attributes' ordering.
 - Can manipulate the values.
- Syntax: $\pi_A(R)$
 - R is input
 - A is list of projection expressions (attribute list)
 - The attribute-list A contains the list of attributes in relation R that will be selected.
- Semantics:
 - Select out only the attribute values given in the attribute-list A from all tuples in relation R
 - Set: $\{ t.A \mid t \in R \}$
 - Bag: $\{ (t.A)^n \mid t^n \in R \}$
 - process each tuple independently; a tuple may appear in the resulting relation multiple times

Projection: Example

- $\pi_b(R)$

R

a	b
1	13
3	12
6	14

Result

b
13
12
14

Compose: Select and Project

- to pick both columns and rows, we can compose operators
- $\pi_{A_1, A_2, \dots, A_n}(\sigma_{\text{condition}}(\text{Expression}))$

Renaming

- To unify schemas for set operators
- For disambiguation in “self-joins”
- Syntax: $\rho_A(R)$
 - R is input
 - A is list of attribute renaming $b \leftarrow a$
- Semantics:
 - Applies renaming from A to inputs
 - Set: $\{ t.A \mid t \in R \}$
 - Bag: $\{ (t.A)^n \mid t^n \in R \}$

Renaming: Example

- $\rho_{c \leftarrow a}(\mathbf{R})$

R

a	b
1	13
3	12
6	14

Result

c	b
1	13
3	12
6	14

Cross Product

- Combine two relations (a.k.a Cartesian product/Cross Join)
- Generate a relation that contains all possible combinations of tuples from the input relations.
- **Syntax:** $R \times S$
 - R and S are inputs
- **Semantics:**
 - All combinations of tuples from R and S
 - = mathematical definition of **cross product**
 - **Set:** $\{ (t,s) \mid t \in R \text{ AND } s \in S \}$
 - **Bag:** $\{ (t,s)^{n \times m} \mid t^n \in R \text{ AND } s^m \in S \}$
- The cartesian product will produce many meaningless results

Cross Product: Example

• $R \times S$

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
1	13	a	5
1	13	b	3
1	13	c	4
3	12	a	5
3	12	b	3
3	12	c	4

Join Types

- **Cross Join**
 - Same thing as Cartesian product.
- **Inner Join**
 - Each tuple in the first relation must have a corresponding match in the second relation.
 - **Natural Join**, **Theta/Equi Join**, **Semi Join**, **Anti Join**.
- **Outer Join**
 - Each tuple in the one relation does not need to have a corresponding match in the other relation.

Condition/Theta Join operation

- Combine related tuples from two relations
- Match tuples using some arbitrary join predicate defined by θ
 - If θ is just an equality predicate, then it is called an [Equi Join](#).
- Syntax: $R \bowtie_{\theta} S$
 - R and S are inputs
 - θ is a condition
- Semantics:
 - All combinations of tuples from R and S that match θ
 - Set: $\{ (t,s) \mid t \in R \text{ AND } s \in S \text{ AND } (t,s) \text{ matches } \theta \}$
 - Bag: $\{ (t,s)^{n \times m} \mid t^n \in R \text{ AND } s^m \in S \text{ AND } (t,s) \text{ matches } \theta \}$
- The condition join operation is a shorthand notation for the following
 - $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

Condition/Theta Join: Example

• $R \bowtie_{a=d} S$

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
3	12	b	3

Natural Join

- Generate a relation that contains all tuples that are a combination of two tuples (one from each input relation) with a common value(s) for one or more attributes.
 - Enforce equality on all attributes with same name (The name and type of the attribute must be same).
- Eliminate one copy of duplicate attributes
- Syntax: $R \bowtie S$
 - R and S are inputs
- Semantics:
 - All combinations of tuples from R and S that match on common attributes
 - A = common attributes of R and S
 - B = exclusive attributes of S
 - Set: $\{(t, s.B) \mid t \in R \text{ AND } s \in S \text{ AND } t.A = s.A\}$
 - Bag: $\{(t, s.B)^{n \times m} \mid t^n \in R \text{ AND } s^m \in S \text{ AND } t.A = s.A\}$
- The **natural join** operation is a shorthand notation for the following
 - $R \bowtie S = \pi_L(\sigma_{R.A=S.A}(R \times S))$, where $L = (\text{list of attributes in } R) \cup B$

Natural Join: Example

• $R \bowtie S$

R

a	b
1	13
3	12

S

c	a
a	5
b	3
c	4

Result

a	b	c
3	12	b

Left-outer Join

- **Outer Join**: Retain information that would have been lost from the relations, replacing missing data with **NULLs**
- **Left-outer Join**: keep data from the left-hand relation
- **Syntax**: $R \bowtie_c S$
 - R and S are inputs
 - c is condition
- **Semantics**:
 - Contains all tuples in R join S
 - Also includes every tuple in R that is not joined with a tuple in S , after padding a **NULL** ($t \in R$ without match, fill S attributes with **NULL**)

$\{(t,s) \mid t \in R \text{ AND } s \in S \text{ matches } c\}$
union
 $\{(t, \text{NULL}(S)) \mid t \in R \text{ AND NOT exists } s \in S: (t,s) \text{ matches } c\}$

Left-outer Join: Example

- $R \bowtie_{a=d} S$

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
1	13	NULL	NULL
3	12	b	3

Right-outer Join

- **Right-outer Join**: keep data from the right-hand relation
- **Syntax**: $R \bowtie_c S$
 - R and S are inputs
 - c is condition
- **Semantics**:
 - $R \text{ join } S$
 - $s \in S$ without match, fill R attributes with $NULL$

$\{(t,s) \mid t \in R \text{ AND } s \in S \text{ matches } c\}$
 union
 $\{(NULL(R),s) \mid s \in S \text{ AND NOT exists } t \in R: (t,s) \text{ matches } c\}$

Right-outer Join: Example

- $R \bowtie_{a=d} S$

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
NULL	NULL	a	5
3	12	b	3
NULL	NULL	c	4

Full-outer Join

- **Full-outer Join**: keep data from both relations

- **Syntax**: $R \bowtie_c S$

- R and S are inputs
- c is condition

- **Semantics**:

$\{(t,s) \mid t \in R \text{ AND } s \in S \text{ AND } (t,s) \text{ matches } c\}$

union

$\{(\text{NULL}(R),s) \mid s \in S \text{ AND NOT exists } t \in R: (t,s) \text{ matches } c\}$

union

$\{(t,\text{NULL}(S)) \mid t \in R \text{ AND NOT exists } s \in S: (t,s) \text{ matches } c\}$

Full-outer Join: Example

- $R \bowtie_{a=d} S$

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
1	13	NULL	NULL
NULL	NULL	a	5
3	12	b	3
NULL	NULL	c	4

Aggregation

- Operators that summarize or aggregate the values in a single attribute of a relation
- Operators are the same in relational algebra and SQL.
- All operators treat a relation as a **bag** of tuples.
- **SUM**: computes the sum of a column with numerical values.
- **AVG**: computes the average of a column with numerical values.
- **MIN** and **MAX**:
 - for a column with numerical values, computes the smallest or largest value, respectively.
 - for a column with string or character values, computes the lexicographically smallest or largest values, respectively
- **COUNT**: computes the number of **non-NULL** tuples in a column.
 - In SQL, can use **COUNT (*)** to count the number of tuples in a relation
 - **Null** values counted by **COUNT(*)**, discarded by other aggregate operators.

Aggregation

- Grouping and aggregation generally need to be implemented and optimized together
- Syntax: $\sigma_G \gamma_A(R)$
 - A is list of aggregation functions
 - G is list of group by attributes
- Semantics:
 - Build groups of tuples according G and compute the aggregation functions from each group
 - $\{t.G, \text{agg}(G(t)) \mid t \in R\}$
 - $G(t) = \{t' \mid t' \in R \text{ AND } t'.G = t.G\}$

Aggregation: Example

- $b\gamma_{sum(a)}(R)$

R

a	b
1	1
3	1
6	2
3	2

Result

sum(a)	b
4	1
9	2

Duplicate Removal/Elimination

- **Syntax:** $\delta(R)$: is the relation containing exactly one copy of each tuple in R
 - R is input
- **Semantics:**
 - Remove duplicates from input
 - **Set:** N/A
 - **Bag:** $\{t^1 \mid t^n \in R\}$
- **Duplicate elimination** is expensive, since tuples must be sorted or partitioned.

Duplicate Removal

- $\delta(R)$

R

a	b
1	13
1	13
6	14

Result

a	b
1	13
6	14

Union, Intersection, and Difference

- Union, intersection, and difference need new definitions for bags.
- Input: R and S
 - Have to have the same schema
 - Union compatible: two relations have the same schema: exactly same attributes drawn from the same domain

- Syntax: $R \cup S$
 - Generate a relation that contains all tuples that appear in either only one or both input relations.
 - R and S are **union-compatible** inputs
- Semantics:
 - Set: $\{t \mid t \in R \text{ OR } t \in S\}$
 - Bag:
 - An element appears in the **union** of two bags the **sum** of the number of times it appears in each bag.
 - $\{(t,s)^{n+m} \mid t^n \in R \text{ AND } s^m \in S\}$
 - e.g., $\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$

Union: Example

• $R \cup S$

R

a
1
3

S

b
1
2
3

Result

a
1
2
3
1
3

Intersection

- Syntax: $R \cap S$
 - Generate a relation that contains only the tuples that appear in both of the input relations
 - R and S are **union-compatible** inputs
- Semantics:
 - Set: $\{t \mid t \in R \text{ AND } t \in S\}$
 - Bag: $\{(t,s)^{\min(n,m)} \mid t^n \in R \text{ AND } s^m \in S\}$
 - An element appears in the **intersection** of two bags the **minimum** of the number of times it appears in either
 - $\{1,2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\}$

Intersection: Example

• $R \cap S$

R

a
1
3

S

b
1
2
3

Result

a
1
3

Set Difference

- Syntax: $R - S$
 - Generate a relation that contains only the tuples that appear in the first and not the second of the input relations.
 - R and S are **union-compatible** inputs
- Semantics:
 - Set: $\{t \mid t \in R \text{ AND NOT } t \in S\}$
 - Bag: $\{(t,s)^{\max\{0,n-m\}} \mid t^n \in R \text{ AND } s^m \in S\}$
 - An element appears in the **difference** $R - S$ of bags as many times as it appears in R , **minus** the number of times it appears in S .
 - *But never less than 0 times.*
 - $\{1,2,1,1\} - \{1,2,3\} = \{1,1\}$

Set Difference: Example

- $R - S$

R

a
1
5

S

b
1
2
3

Result

a
5

- Logical Query Plan (LQP) Generation
 - Converting a Parse Tree into an initial logical query plan (tree)

Conversion into Relational Algebra

- When the query is expressed as a valid **parse tree**, we can generate a **LQP** expressed by **relational algebra operators**
- **SFW without sub-queries**:
 - replace the relations in the **<FromList>** by the **product**, \times , of all relations
 - this **product** is the argument of a **selection**, σ_C , where **C** is the **<Condition>** expression being replaced
 - this **selection** is in turn the argument of a **projection**, π_L , where **L** is the list of attributes in the **<SelList>**

Canonical Translation to Relational Algebra

- TEXTBOOK version of conversion
- Given an SQL query
- Return an equivalent relational algebra expression

Canonical Translation to Relational Algebra

- **FROM** clause into joins and crossproducts
- **WHERE** clause into selection
- **SELECT** clause into projection and renaming
 - If it has aggregation functions use aggregation
 - **DISTINCT** into duplicate removal
- **GROUP BY** clause into aggregation
- **HAVING** clause into selection
- **ORDER BY** - no counter-part
 - Since a relation is a set (or a bag), there is no ordering defined for a relation. That is, two relations are the same if they contain the same tuples, irrespective of ordering.

Set Operations

- **UNION ALL** into union
- **UNION** duplicate removal over union
- **INTERSECT ALL** into intersection
- **INTERSECT** add duplicate removal
- **EXCEPT ALL** into set difference
- **EXCEPT** apply duplicate removal to inputs and then apply set difference

Logical and physical query plans

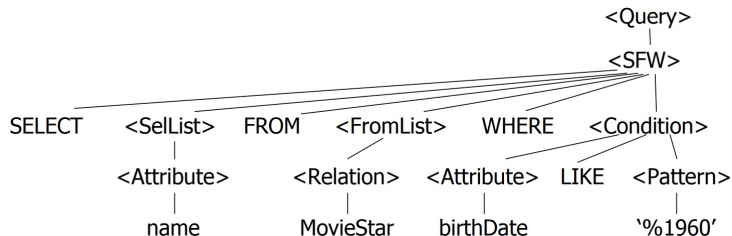
- Both are trees representing query evaluation
- Leaves represent data
- Internal nodes are operators over the data
- Logical plan is higher-level and algebraic
- Physical plan is lower-level and operational

Expression Trees

- Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand
- Leaves are operands
 - Interior nodes are operators, applied to their child or children.

Conversion into Relational Algebra: Example 1

```
SELECT name FROM MovieStar WHERE birthdate LIKE '%1960';
```

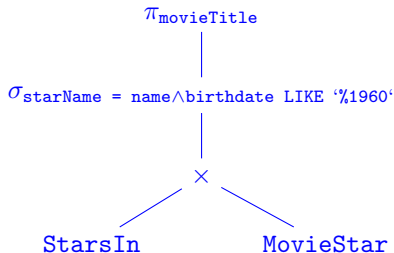


- product of relations in <FromList>
- select tuples using expression in <Condition>
- project wanted attributes in the <SelList>

π_{name}
|
 $\sigma_{\text{birthdate LIKE '%1960'}}$
|
MovieStar

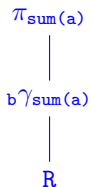
Conversion into Relational Algebra: Example 2

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



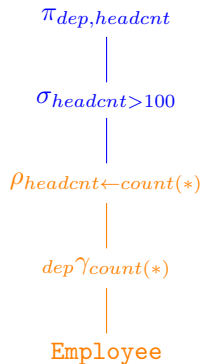
Conversion into Relational Algebra: Example 3

```
SELECT sum(a)  
FROM R  
GROUP BY b
```



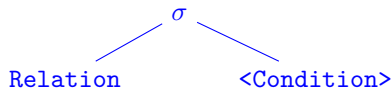
Conversion into Relational Algebra: Example 4

```
SELECT dep, headcnt
FROM (SELECT count(*) AS headcnt, dep
      FROM Employee
      GROUP BY dep)
WHERE headcnt > 100
```



Conversion into Relational Algebra

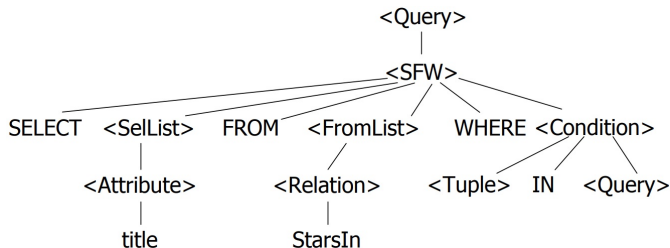
- If we have **sub-queries**, we must remove them by using an intermediate operator - two argument select σ :



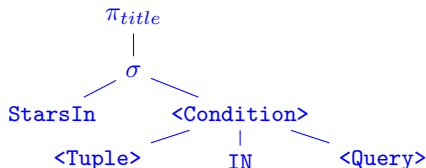
- left child represents relation upon which the selection is performed
- right child is an expression for the condition applied to each tuple of the relation

Conversion into Relational Algebra: Example 5

```
SELECT title FROM StarsIn WHERE starName IN (<Query>;
```



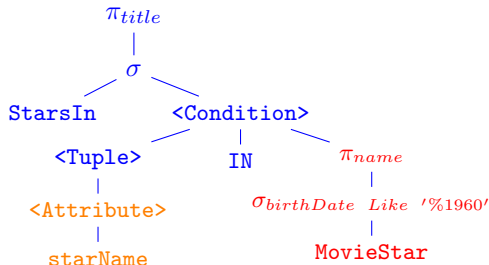
- product of relations in `<FromList>`
- select tuples using expression in `<Condition>`, but use the two-argument select on sub-query
- project wanted attributes in the `<SelList>`



Conversion into Relational Algebra: Example 5(Cont.)

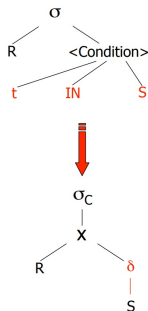
```
SELECT title FROM StarsIn WHERE starName IN (<Query>;
```

- $\langle \text{Tuple} \rangle$ is represented by $\langle \text{Attribute} \rangle - \text{starName}$
- the sub-query $\langle \text{Query} \rangle$ is the query we converted earlier
- This tree needs further transformation



Conversion into Relational Algebra

- Replace two-argument select:
 - different conditions require different rules
 - we will look at $t \text{ IN } S$:
 - replace $\langle \text{Condition} \rangle$ with the tree representing S . If S may have duplicates we must include a δ -operator at the top
 - replace the two-argument selection by a one-argument selection σ_C , where C is the condition that equates each component of tuple t to the corresponding attribute in S
 - give σ_C an argument that is the **product** of R and S

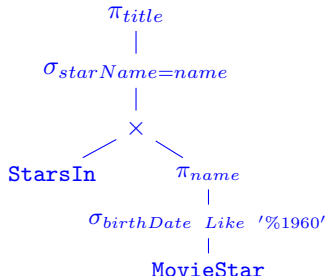
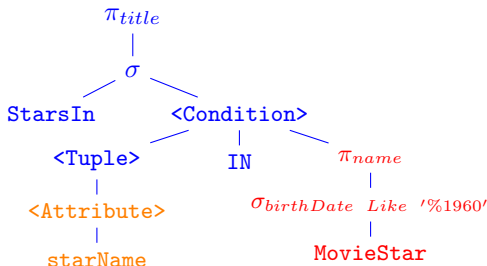


Conversion into Relational Algebra: Example 5(Cont.)

- Example (cont.):

```
SELECT title FROM StarsIn WHERE starName IN (...);
```

- replace **<Condition>** with the tree representing the sub-query
- replace the **two-argument selection** by a **one-argument selection** σ_C , where **C** is *starName = name*
- give σ_C an argument that is the **product** of StarsIn and MovieStar



Conversion into Relational Algebra

- Translating sub-queries can be more complex if the sub-query is correlated to values defined outside its scope
 - we must produce a relation with some extra attributes for comparison with external attributes
 - the extra attributes are later removed using projections
 - any duplicate tuples must be removed
- Translating the parse tree into expressions in algebra may give several equivalent LQP using different operators or just changing the order of the operators

Summary

- SQL text \rightarrow Internal representation
- Semantic checks
- View unfolding
- Conversion into Relational Algebra

Next

- When we have translated the parse tree to a relational algebra expression, the next step is to optimize the expression