# Databases 2

# Lecture V

Alessandro Artale

Faculty of Computer Science – Free University of Bolzano

Room: 221

artale@inf.unibz.it

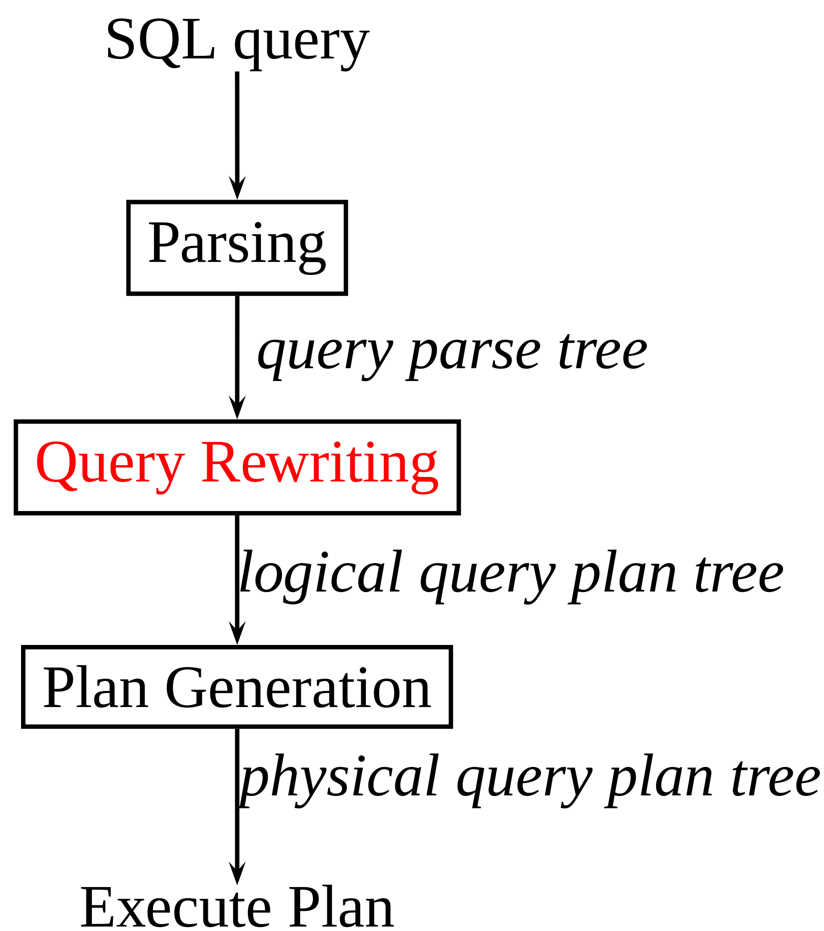
http://www.inf.unibz.it/∼artale/

2003/2004 – First Semester

## Summary of Lecture V

- Query Rewriting;
  - Algebraic Transformations;

# Query Compilation Overview



# Query Rewriting Phase

Three steps process:

1. Taking the parse tree of the query it generates a first logical query plan (i.e., an equivalent relational algebra expression tree);
2. Generate other equivalent logical query plans (adopting algebraic transformations);
3. Choose the best logical query plan based on cost estimation functions.

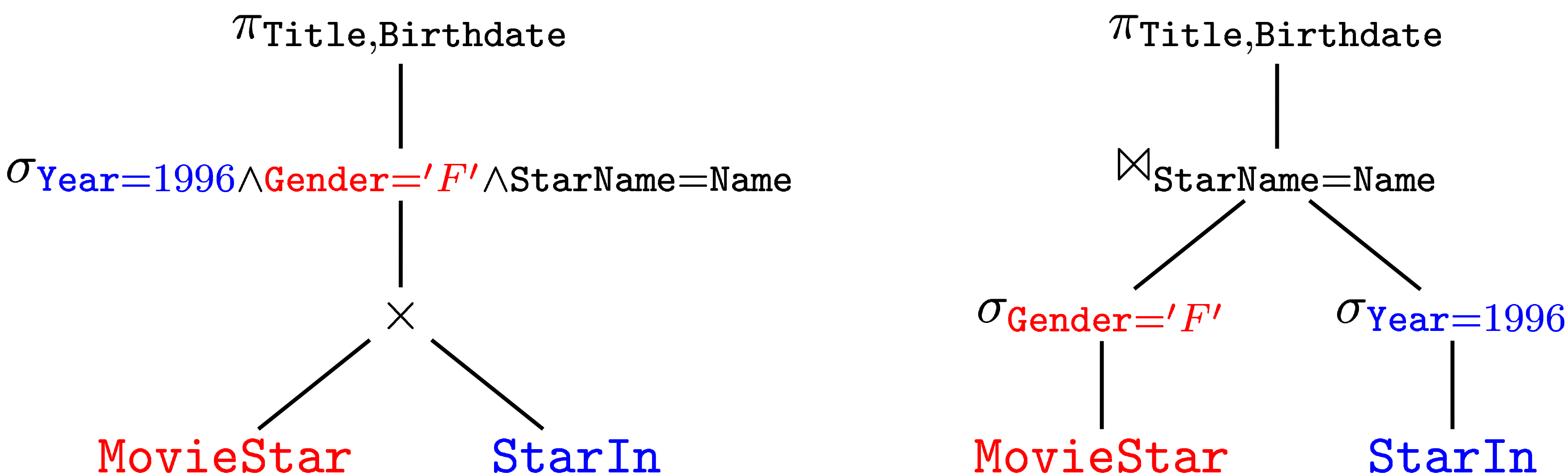
# Example

$\texttt{MovieStar}(\texttt{Name}, \texttt{Address}, \texttt{Gender}, \texttt{Birthdate})$

$\texttt{StarIn}(\texttt{Title}, \texttt{Year}, \texttt{StarName})$

**Query:** *"Find the movie title and the birthdate for those female stars who appeared in movies in* 1996".

**SELECT** Title, Birthdate
**FROM** MovieStar, StarIn
**WHERE** Year=1996 **AND** Gender='F' **AND** Name=StarName;



# Algebraic Transformations

Laws that turn an expression tree into an equivalent one that may have a more efficient physical query plan.

- **Commutative and Associative Laws.**
  - **Commutative Law.** The order of the operands doesn't matter:

$$x + y = y + x; \quad x - y \neq y - x$$

  - **Associative Law.** The order in which you perform the operations doesn't matter:

$$x + y + z = (x + y) + z = x + (y + z); \quad x - y - z \neq x - (y - z)$$

# Algebraic Transformations (cont.)

- Several relational algebra operators are both associative and commutative:

  $R \times S = S \times R; \quad (R \times S) \times T = R \times (S \times T)$

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

  $R \cup S = S \cup R; \quad (R \cup S) \cup T = R \cup (S \cup T)$

  $R \cap S = S \cap R; \quad (R \cap S) \cap T = R \cap (S \cap T)$

- **Note:** For theta-join the associative law does not hold. Consider the relations $R(A, B); S(B, C); T(C, D)$ then:

  $$(R \bowtie_{R.B>S.B} S) \bowtie_{A<D} T \neq R \bowtie_{R.B>S.B} (S \bowtie_{A<D} T)$$

  The latter join doesn't even make sense, because $A$ is neither an attribute of $S$ nor of $T$ .

# Laws Involving Selection

Selections tend to reduce the size of relations: One of the most effective optimization rules is to *push selections down* the expression tree as far as it preserves equivalence.

- *Order Swapping.*
  - $\sigma_{C_1}(\sigma_{C_2}(R)) = \sigma_{C_2}(\sigma_{C_1}(R))$
- *Splitting law.*
  - $\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$
  - $\sigma_{C_1 \vee C_2}(R) = (\sigma_{C_1}(R)) \cup_S (\sigma_{C_2}(R))$, $R$ must be a set.
- *Pushing Selection Down Union and Difference.*
  - $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
  - $\sigma_C(R - S) = \sigma_C(R) - S = \sigma_C(R) - \sigma_C(S)$

# Laws Involving Selection Cont.)

- *Pushing Selection Down Join, Theta Join, Product and Intersection*.
  - $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$, If $R$ has all the attributes mentioned in $C$;
  - $\sigma_C(R \bowtie S) = R \bowtie \sigma_C(S)$, If $S$ has all the attributes mentioned in $C$;
  - $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie \sigma_C(S)$, If both $R$ and $S$ have all the attributes mentioned in $C$ (This case doesn't apply for $\times, \bowtie_C$ because no sharing attributes).

# Laws Involving Selection: Examples

- **Example 1.** Relations: $R(A, B); S(B, C)$. Compute: $\sigma_{A=1}(R \bowtie S)$ by *pushing the selection* down to $R$: $(\sigma_{A=1}(R)) \bowtie S$.

- **Example 2.** Using views (pre-computed queries) has been recently discovered that a good strategy is to first move *up* the selection and then *down* along all possible branches.

  $\mathtt{StarsIn}(\mathtt{title}, \mathtt{year}, \mathtt{starName})$

  $\mathtt{Movie}(\mathtt{title}, \mathtt{year}, \mathtt{studioName})$

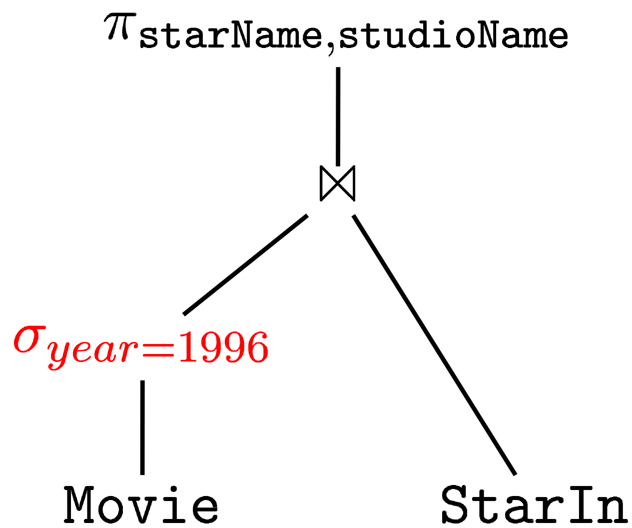
  ```
  CREATE VIEW MovieOf1996 AS
  SELECT      *
  FROM        Movie
  WHERE       year = 1996;
  ```

  We can ask the query: "*which stars worked for which studio in 1996*?":

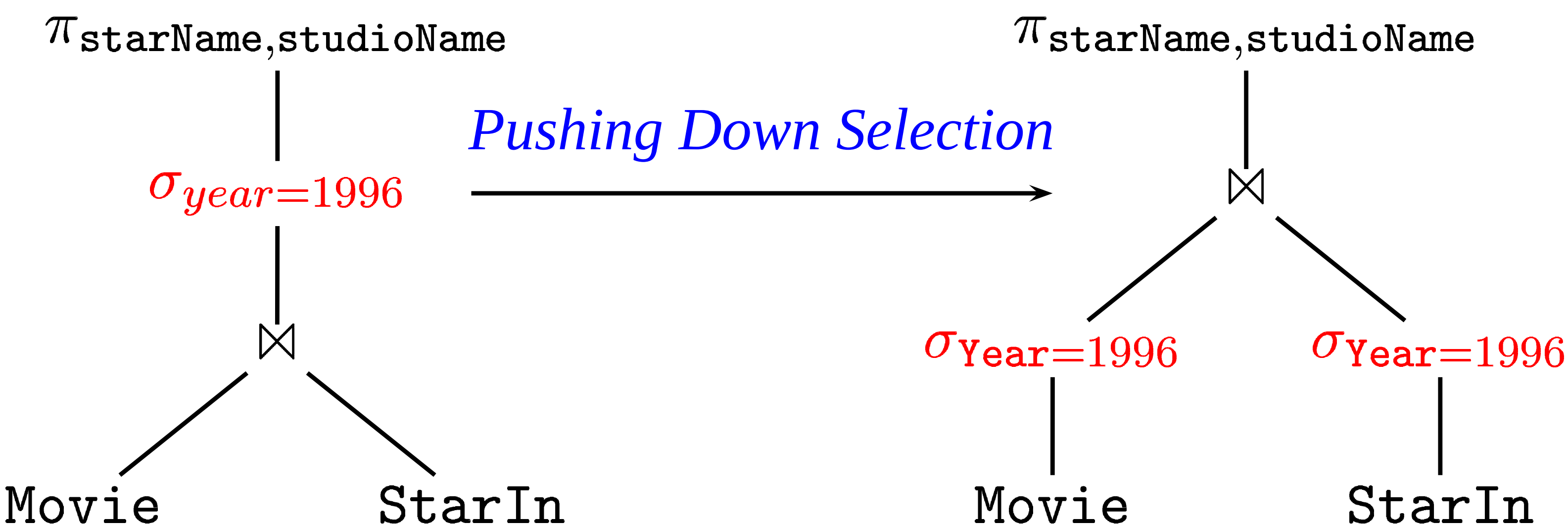
  ```
  SELECT starName, studioName
  FROM   MovieOf1996 NATURAL JOIN StarsIn;
  ```

# Example 2 (cont.)

The query is converted in the following logical query plan:




We apply "backwards" the rule: $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$:



# Pushing Projection

Pushing down projections reduces the number of attributes in intermediate results – thus their size is reduced, too.

- $\pi_L(R \times S) = \pi_{A_1,\ldots,A_n}(R) \times \pi_{B_1,\ldots,B_m}(S)$
  with $L = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$.

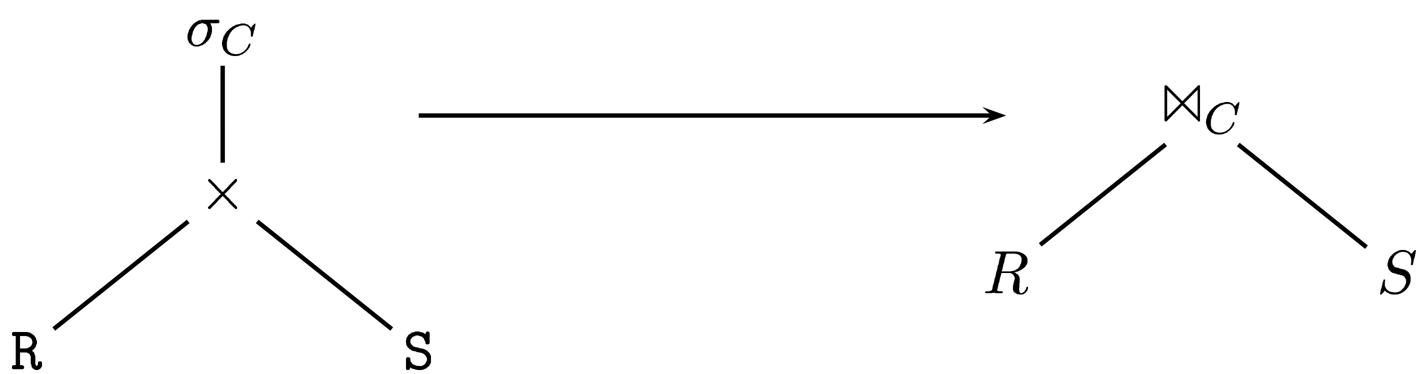
- $\pi_L(R \bowtie S) = \pi_L(\pi_{A_1,\ldots,A_n,A_{n+1},\ldots,A_{n+k}}(R) \bowtie \pi_{B_1,\ldots,B_m,B_{m+1},\ldots,B_{m+p}}(S))$
  where $A_{n+1}, \ldots, A_{n+k}, B_{m+1}, \ldots, B_{m+p}$ are involved in the natural join but not in the external projection.

- $\pi_L(R \bowtie_C S) = \pi_L(\pi_{A_1,\ldots,A_n,A_{n+1},\ldots,A_{n+k}}(R) \bowtie_C \pi_{B_1,\ldots,B_m,B_{m+1},\ldots,B_{m+p}}(S))$
  where $A_{n+1}, \ldots, A_{n+k}, B_{m+1}, \ldots, B_{m+p}$ are involved in the join condition $C$ but not in the external projection.

# Pushing Projection (cont.)

- Projection *cannot* be pushed down set operators (except bag union). Example, let $R(A, B) = \{(1, 2)\}$ and $S(A, B) = \{(1, 3)\}$. Then, $\pi_A(R \cap S) = \emptyset$, but $\pi_A(R) \cap \pi_A(S) = \{(1)\} \cap \{(1)\} = \{(1)\}$.
- $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$, where $M = L \cup C$.
  If $C \subseteq L$ then the two operations commute:
  $\pi_L(\sigma_C(R)) = \sigma_C(\pi_L(R))$.
  **Note.** If $R$ is a stored relation with an index on the selection attributes, push down the selection.

# Law between Join and Product




$$\boxed{\sigma_C(R \times S) = R \bowtie_C S}$$

- Products followed by Selections are substituted by Joins since algorithms for computing Joins are more efficient.

## Laws Involving Duplicate Elimination $\delta$

- **General goal:** Duplicate Elimination is an expensive operation, and moving it around we can:

  1. Eliminate it altogether when it meets a (*set*) union, intersection or difference; or a group-by (which always produces a set); or a stored relation.

$$
\begin{aligned}
\delta(R \cup_S S) &= R \cup_S S \\
\delta(R \cap_S S) &= R \cap_S S \\
\delta(R -_S S) &= R -_S S \\
\delta(\gamma_L(R)) &= \gamma_L(R) \\
\delta(R) &= R, \qquad \text{if } R \text{ is a stored relation.}
\end{aligned}
$$

## Laws Involving Duplicate Elimination $\delta$ (cont.)

- Pushing $\delta$ down the tree reduces the size of intermediate relations.

$$
\begin{array}{lcl}
\delta(R \times S) & = & \delta(R) \times \delta(S) \\
\delta(R \bowtie S) & = & \delta(R) \bowtie \delta(S) \\
\delta(R \bowtie_C S) & = & \delta(R) \bowtie_C \delta(S) \\
\delta(\sigma_C(R)) & = & \sigma_C(\delta(R)) \\
\delta(R \cap_B S) & = & \delta(R) \cap_B S = R \cap_B \delta(S) = \delta(R) \cap_B \delta(S)
\end{array}
$$

## Laws Involving Duplicate Elimination $\delta$ (cont.)

- $\delta$ **cannot** be pushed down $\cup_B, -_B, \pi$.

  **Example.** $R(A, B) = \{(1, 2); (1, 3)\}$.

$$\begin{array}{lcl}
\delta(\pi_A(R)) & = & \delta(\{(1); (1)\}) & = & \{(1)\} \\
\pi_A(\delta(R)) & = & \pi_A(R) & = & \{(1); (1)\}
\end{array}$$

For a counter example for $\cup_B$ consider a common tuple; while for $-_B$ $R$ with 2 copies of a tuple $t$ and $S$ with just one occurrence of $t$.

## Selecting the Best Executable Plan

- Starting with the SQL query definition an equivalent relational algebra expression is obtained;
- Apply algebraic transformations to find other possibly better plans;
- Generate physical query plans for each logical plan by choosing an order and a grouping for the associative and commutative operations, and by choosing an algorithm for each operator;
- Evaluate the cost of each physical plan, using estimates of sizes for intermediate results, possibly using statistics about the stored relations.