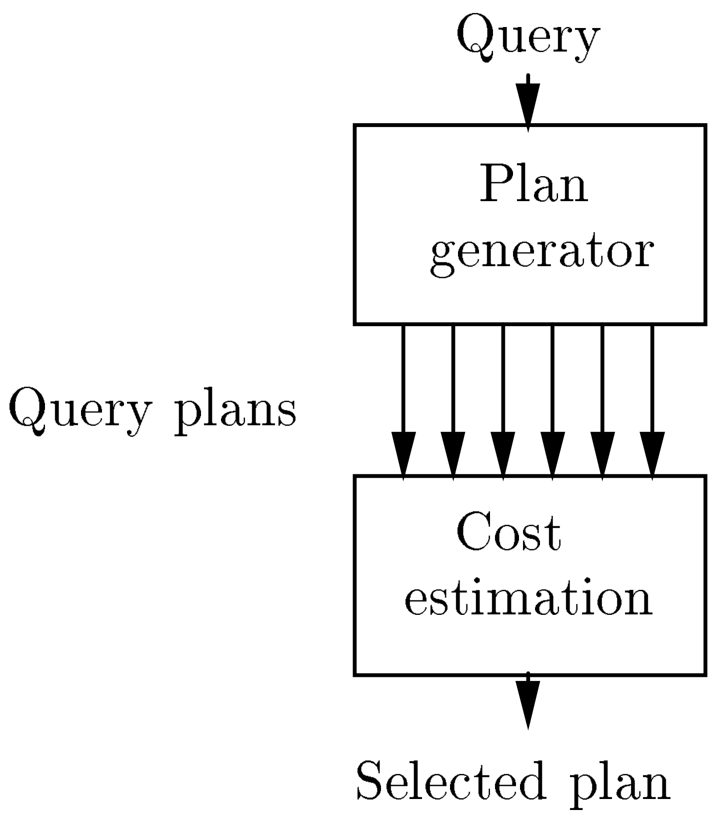
Today:

- Query optimization.
- Algebraic laws; extensions to relational algebra for select-distinct, grouping.

Soon:

- Estimating costs.
- Algorithms for computing joins, other operations.

Query Optimization



Query Plan

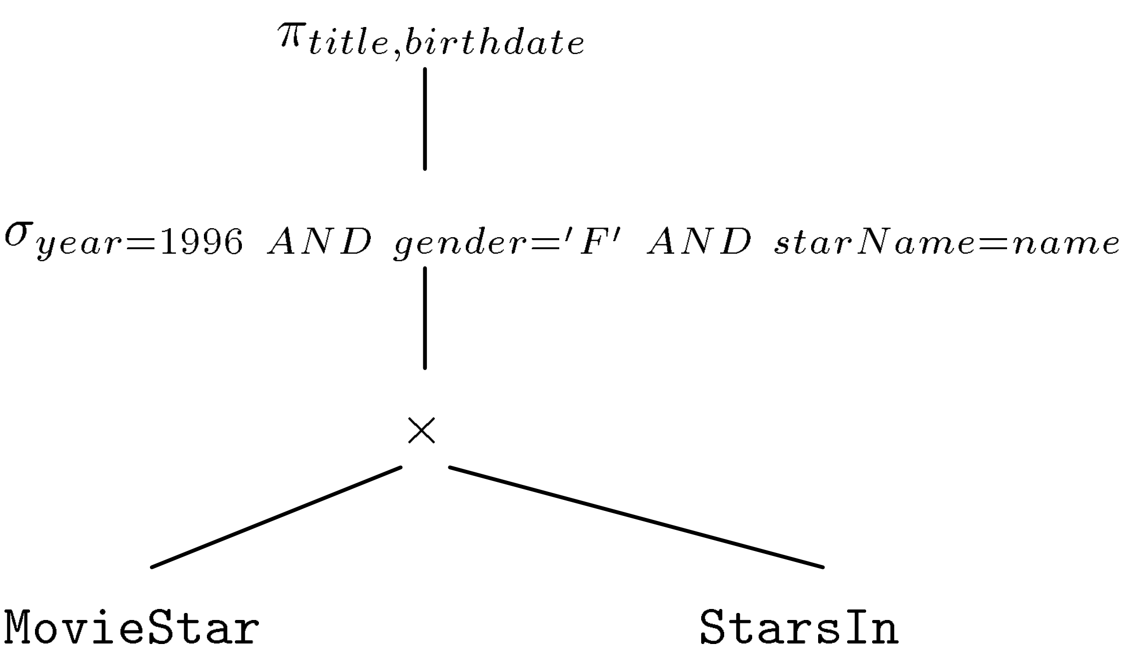
- Choose operations, e.g., σ, $\bowtie$.
- Order operations.
- Detailed strategy of operations, e.g.:
    - Join method.
    - *Pipelining*: consume result of one operation by another, to avoid temporary storage on disk.
    - Use of indexes?
    - Sort intermediate results?

Example

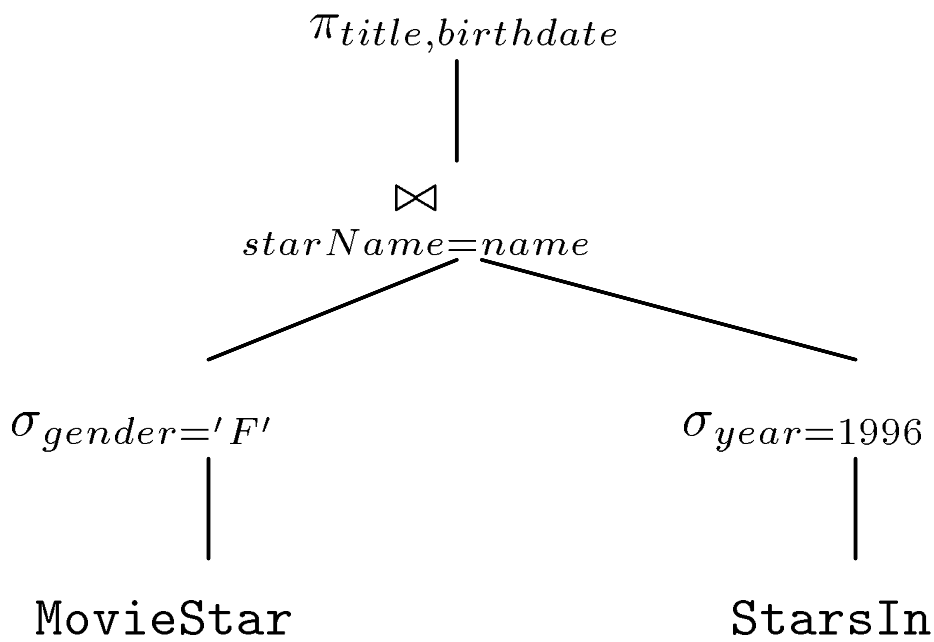
```
MovieStar(name, addr, gender, birthdate)
StarsIn(title, year, starName)
```

```
SELECT title, birthdate
FROM MovieStar, StarsIn
WHERE year = 1997 AND
    gender = 'F' AND
    starName = name;
```

Plan I (from definition)



Plan II



- Join method?
- Can we pipeline the result of one or both selections, and avoid storing the result on disk temporarily?
- Are there indexes on `MovieStar.gender` and/or `StarsIn.year` that will make the σ's efficient?

Generating Plans

- Start with query definition.
  - A plan, but usually a terrible one.
- Apply algebraic transformations to find other plans.
  - Usually, there is a preferred direction.
  - Relational algebra is a good start, but we need also to consider: `GROUP BY`, duplicate elimination, `HAVING`, `ORDER BY`.
- Evaluate the cost of each generated plan, using estimates of sizes for intermediate results, possibly using statistics about the stored relations.

Algebraic Transformations

Laws give *equivalent* expressions. meaning that whatever relations are substituted for variables, the results are the same.

- Commutative and associative laws.
  - ❖ Example: for natural join: $R \bowtie S = S \bowtie R$; $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$.
  - ❖ Leads to *join-ordering problem* — important for complex queries.
  - ❖ Same idea for $\times$, $\cup$, $\cap$.
- But beware theta-join; associative law does not hold.
  - ❖ Example: relations $R(a,b)$, $S(b,c)$, $T(c,d)$;

$$(R \underset{R.b>S.b}{\bowtie} S) \underset{a<d}{\bowtie} T \neq R \underset{R.b>S.b}{\bowtie} (S \underset{a<d}{\bowtie} T)$$

The latter doesn't even make sense, because a is not an attribute of S or T.

Laws Involving Selection

- Splitting:
  - ✦ $\sigma_{C_1 \ AND \ C_2}(R) = \sigma_{C_1}\big(\sigma_{C_2}(R)\big)$
  - ✦ $\sigma_{C_1 \ OR \ C_2}(R) = \sigma_{C_1}(R) \cup \sigma_{C_2}(R)$
- "Pushing selections":
  - ✦ $\sigma_C(R \bowtie S) = \big(\sigma_C(R)\big) \bowtie S$, as long as condition C makes sense on R.
  - ✦ Also possible to move σ_C to S if C makes sense there.
  - ✦ We can even move σ_C to *both* if it makes sense.
  - ✦ Same ideas for commuting σ with $\times$, $\underset{C}{\bowtie}$.
- Selection and union, intersection, difference:
  - ✦ $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
  - ✦ Similar for $\cap$, $-$.
- Selection and product — combine to form a join:
  - ✦ $\sigma_C(R \times S) = R \underset{C}{\bowtie} S$

Directionality in Selection Pushing

SKS says always push downward.

- Example: relations $R(a,b)$, $S(b,c)$. Replace $\sigma_{a=1}(R \bowtie S)$ by $\big(\sigma_{a=1}(R)\big) \bowtie S$.
  - Big win, because we probably reduce the size of the first join argument by a lot.
- Trivial counterexample: what if S is empty?
- Serious counterexample on next slide.

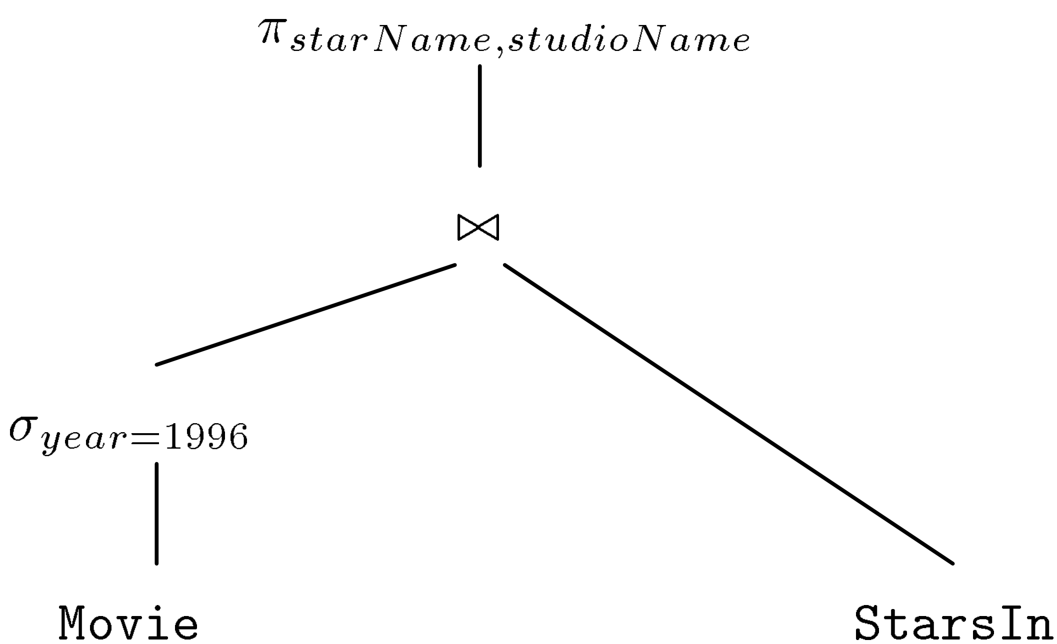
Selections Should Go Up Then Down

StarsIn(title, year, starName)
Movie($\underline{\texttt{title}}$, $\underline{\texttt{year}}$, studioName)

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

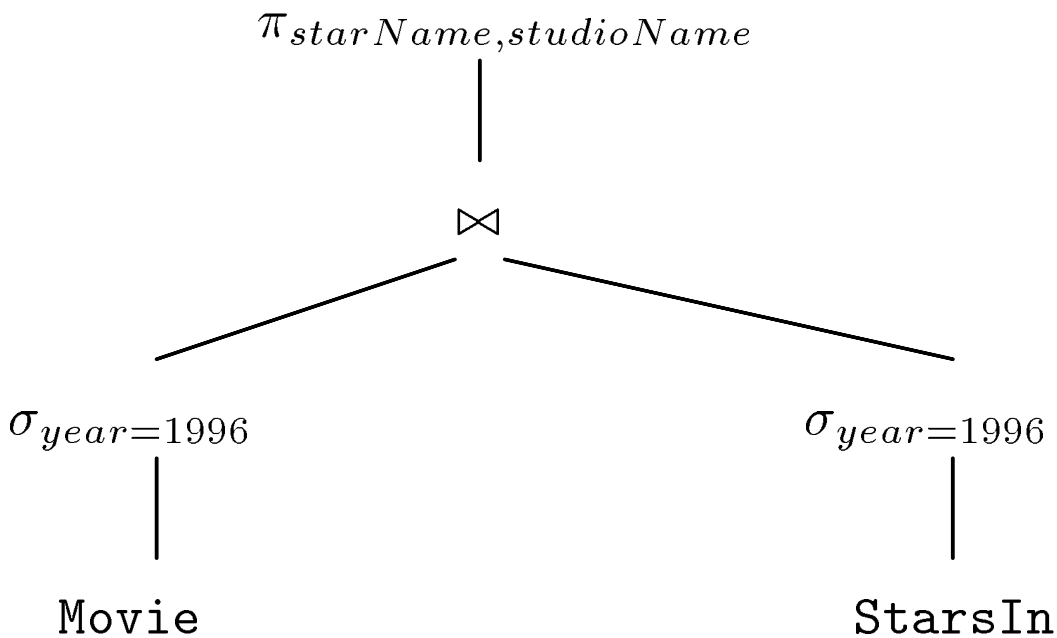
```
SELECT starName, studioName
FROM MoviesOf1996 NATURAL JOIN StarsIn;
```

Initial query:



Probably Better:

Move σ up to root, then down both paths.



Pushing Projections

- $\pi_X(R \bowtie S) = \pi_X\big(\pi_Y(R) \bowtie \pi_Z(S)\big)$, where Y is those attributes of R that are either:
    1. In X, or
    2. A join attribute of R and S.

    ❖ Z defined similarly.
- Similar rules for commuting π with $\times$, $\underset{C}{\bowtie}$, $\cup$.

Problem

Does π commute with $\cap$? With $-$?

Selection and Projection

- $\pi_X\big(\sigma_C(R)\big) \;=\; \pi_X\Big(\sigma_C\big(\pi_Y(R)\big)\Big)$ if Y is X union the attributes mentioned in C.

Should We Push Projections?

SKS says pushing projections down is good, but they are too optimistic. Example:

```
SELECT starName
FROM StarsIn
WHERE year = 1996;
```

- Suppose there is an index on `year`.

Efficient

$$\pi_{starName}$$

$$|$$

$$\sigma_{year=1996}$$

$$|$$

`StarsIn`

Wastes Time

$$\pi_{starName}$$

$$|$$

$$\sigma_{year=1996}$$

$$|$$

$$\pi_{starName,year}$$

$$|$$

`StarsIn`

Operators Outside Relational Algebra

Real query optimizer must deal with:

- Duplicate elimination, and operators that require bag semantics, e.g., `UNION ALL`.
- Group-by and `HAVING`.

Duplicate Elimination

A step in a query tree that involves the relation as a whole.

- We'll use δ as the duplicate-elimination operator, e.g., $\delta(R) = R$ with duplicates eliminated.

Algebraic Laws Involving δ

- Commutes with σ, $\times$, $\bowtie$, $\underset{C}{\bowtie}$, $\cup$, $\cap$, $-$.
  - Examples: $\delta\big(\sigma_{A=c}(R)\big) = \sigma_{A=c}\big(\delta(R)\big)$, $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$.
  - Note that δ goes down *both* paths of a binary operator.
- Remember that $\cup$, etc., eliminate duplicates anyway. Thus, we have rules like: $R \cup S = \delta(R \cup S) = \delta(R) \cup \delta(S)$.
- General goal of moving δ around: it is an expensive operation, and sometimes we can eliminate it altogether when it meets a (set) union, e.g., or a group-by (which always produces a set).

δ and π

Duplicate elimination does not commute with projection.

- Example: $R(A, B) = \{(1,2),\ (1,3)\}$. $\delta\big(\pi_A(R)\big) \neq \pi_A\big(\delta(R)\big)$.

Bag Versions of $\cup$, Etc.

Since SQL allows us to require bag union, etc., we need operators $\cup_B$, $\cap_B$, and $-_B$ to denote these operations.

- **Question**: which of these are valid?
  - $\delta(R \cup_B S) = \delta(R) \cup_B \delta(S)$?
  - $\delta(R \cap_B S) = \delta(R) \cap_B \delta(S)$?
  - $\delta(R -_B S) = \delta(R) -_B \delta(S)$?

Grouping

Introduce operator γ for grouping.

- Takes a list of attributes and aggregated attributes, plus possibly a `HAVING` condition.

Example

`StarsIn(title, year, starName).`

```
SELECT title, MIN(year)
FROM StarsIn
GROUP By title
HAVING COUNT(starName) >= 3
```

$\gamma_{title,MIN(year)|COUNT(starName \geq 3)}(\texttt{StarsIn})$

Laws Involving γ

Not much.

- γ absorbs δ: $\delta\big(\gamma_X(R)\big) = \gamma_X(R)$.
- Some special opportunities, e.g., if the the only aggregation is `MIN` or `MAX`, then we can introduce a δ to apply to the operand relation.
  - Might allow compacting of computation below.