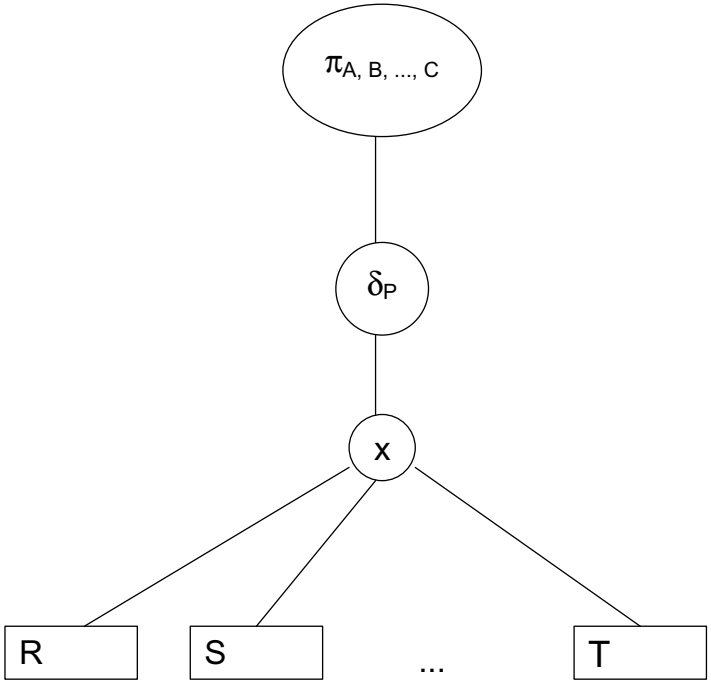
Chapter 3.3 Operator Trees

select A, B, ..., C
from R, S, ..., T
where P

naive Translation:



Optimization Process in principal:

1. Generate naive operator tree

2. Split predicate P into restriction predicate, join pred.

3. Push down restriction predicates

4. Introduce projections and push down

5. estimate size of relations

6. Estimate size of joins:
   number of join-partners per tupel?

7. Estimate cost of Join:
   depending on applicable algorithms.

8. Fix sequnce of joins:
   j! Possibilities for j relations.
   Consider SIPs as side condition

9. Pipelining

Ad. 2: Splitting of predicates

$P \equiv P_R \wedge P_S \wedge ... \wedge P_T \wedge$
$P_{R,S} \wedge P_{R,T} \wedge P_{S,T} \wedge P_{R,S,T}$

Restriction predicate depends only on 1 relation, e.g. P_R

Join-Predicate: depends on several relations, e.g. $P_{R,T}$ specifies join-condition.

Ad. 3: Reduce size of intermediate results:
R. A ≥ 27 ∧ R. A ≤ 30 ∧ R. M# = T. M#

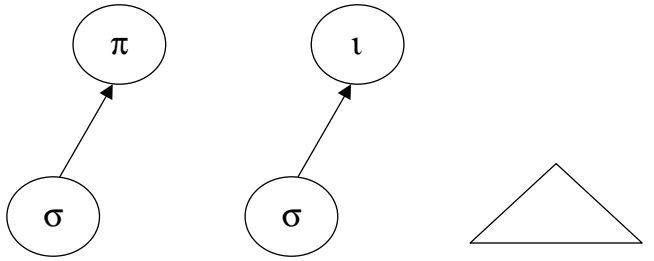
$P_R \equiv$ R. A ≥ 27 ∧ R. A ≤ 30
$P_{R,T} \equiv$ R. M# = T. M#
$P \equiv P_R \wedge P_{R,T}$

corresponding tree?

Ad. 4: Project away those attributes, which are not needed for later joins or for the final results (keep in mind pipelining)

Ad. 6, 7, 8: see later, Join-Algorithms

Ad 9:



Pass on result tuples of σ, do not store!

Example:

Part (P#, S#, QTY, PRICE)

create table Part
(P# **integer**,
S# **smallint**,
QTY **smallint**,
PRICE **real**) **key is** (P#, S#)

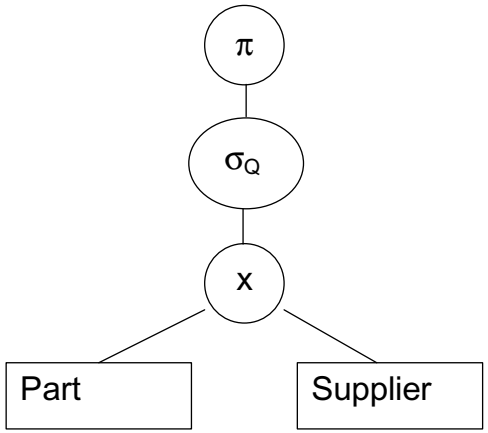
Supplier (S#, S-City, S-Rating)

create table Supplier
(S# **smallint**,
S-City **varchar** (25),
S-Rating **smallint**) **key is** S#

Query 1: P#, S#, City for parts with PRICE > 100 and S-Rating < 5

select P.P#, P.S#, S.S-City
from Part P, Supplier S
where P.PRICE > 100 **and**
S.S-Rating < 5 **and**
P.S# = S.S#

Step 1:



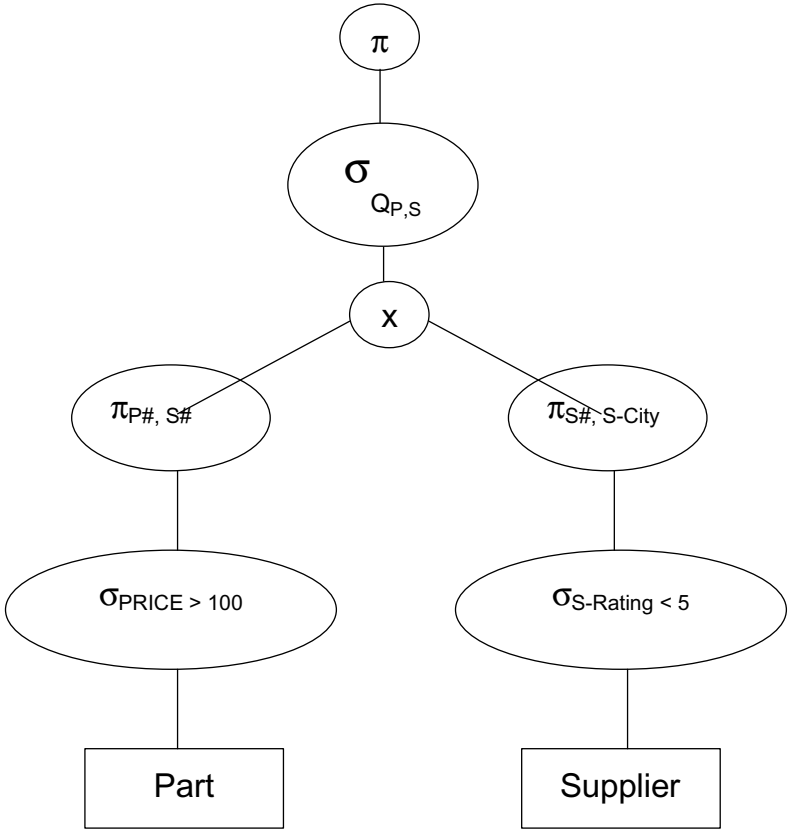
Step 2:

$Q_P \equiv$ P.PRICE > 100
$Q_S \equiv$ S.S-Rating < 5
$Q_{P,S} \equiv$ P.S# = S.S#
$Q \equiv Q_P \wedge Q_S \wedge Q_{P,S}$

Step 3:

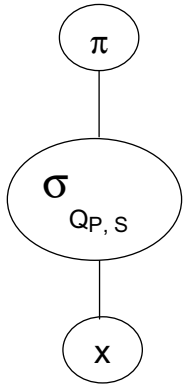
Part	Supplier
P#, S#, QTY, PRICE	S#, S-City, S-Rating

Step 4: Introduce projections



Steps 5, 6, 7, 8: Not shown here

Step 9:



replace by
natural join

Preview: modifiy step 3

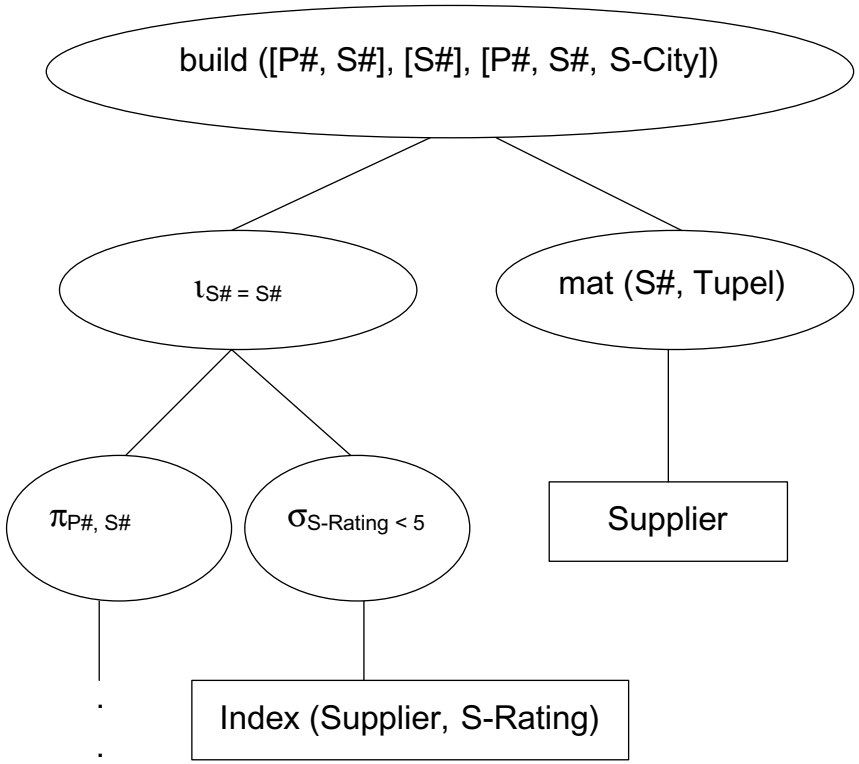
Assumption:secondary index

Index (Supplier, S-Rating)

materializing operation: mat (Rel, TID, Tuple)

Build-Op.: to construct tuple

build ([Input, Par.], [call parameter], [result])



⇒ only really needed tuples are fetched from Supplier

Note 1: Combination of mat with π, if e.g. Supplier is stored on another computer, in order to save communication

Note 2: additional pipelining after step 4?