3.4 Complexity of relational Algorithms

1. Selection/Restriction: let $|R| = n$

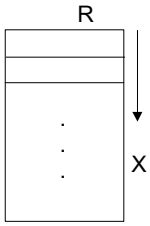
Every variable in predicate must be instantiated with every tuple from R.
A predicate is called selection predicate: only 1 var. X, therefore only one relation is affected, predicate P(X), e. g.
select * from RX **where** P(X)

Note: if there are several variables over the same relation R, we get a "selfjoin" of R with R

1.1 Relation-Scan

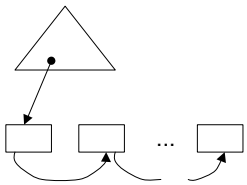
Instantiate var. X with all tuples of R
- order is arbitrary – and evaluate (PX)
  cost **O(n)**

constant depends on details of P, e.g.:
  X.A > 17 **and** X.B = "männlich"
  storage of R
  DBS configuration e.g. Cache size ect.



1.2. B-Tree Index

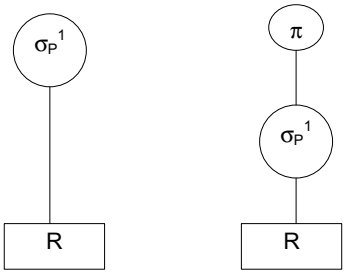
Assumption: Index (R,A) and comparison of constants or comparison with interval, e.g.
P(X) ~ X.A = 17 or
X.A > 10 **and** X.A < 21



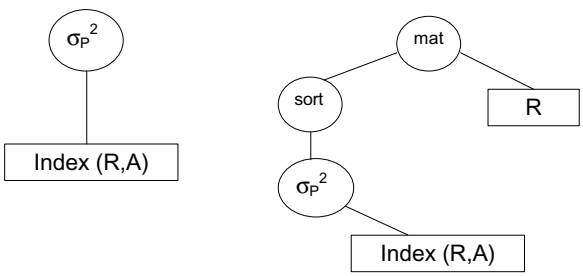
Inverval within Index can be read sequentially, i.e.
(log n) + O(Interval-size)
resp. **O(size of answer)**

comparison with several constants
X.A = 17 or X.A =19
or several intervals
➔ costs are additive

1.2.1 Primary Index: Index-Selection yields set of tuples or {TID}



1.2.2 Secondary Index: yields TID-set KR, or tuples must still be materialized



1.3 Hash-Index

Search for constant comparison: **O(1)**
search for interval: for small intervals, enumeration: **O(Intervalsize)**
Otherwise no support by index, i.e. relation scan required: **O(n)** as in 1.1

2. Projection π

2.1 case 1: primary key or key candidate is part of the result

➔ duplicates cannot arise
➔ with |R| = n we get |πR| = n, also **O(n)**

2.2 case 2: not case 1

➔ duplicate may arise, must be removed!
- use $\overline{\pi}$: **O(n)**
- nested loop: **$O(n^2)$**
- sort first: **O(n log n)**
- hashing: **O(n*k)**

k = cardinality of the largest collision class, only useful, if R fits in AS

Note: k grows with $O(n)$ for fixed size of hash-table:
check of duplicates in every collision class
$|K_i| = k_i$, i = 1,...,l costs

read R: n
write R into AS: %
check duplicates for K_i : k_i^2
write R: n
If all classes have the same size:
n classes * k^2 = n*k
k

3. Join-Algorithms

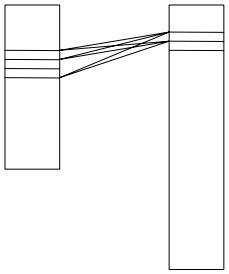
$R \iota S \quad |R| = m \quad |S| = n$

3.1 Nested Loop Join:

$O(m * n) \sim \pi \sigma x$

3.2 Merge Join

R, S suitably sorted, i.e. according to Join-Attribut



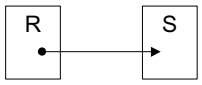
costs:

$\leq |R| + |R| .k + |S|$
if every tuple in R has most k join-partners in S

$\Rightarrow O(n+ m)$

3.3 Sort-Merge-Join:

- first sort R, S
  $O(m \log m) + O(n \log n)$

Note: for 

S is properly sorted

- then merge-join like in 3.2

3.4 Hash-Join: via join-attribute:

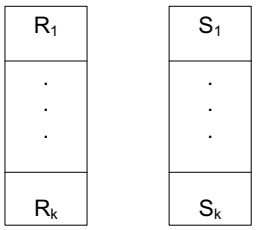
- partition both relations with r. to Hash-Function into classes, i. e.
  $r \equiv r' <> h(r) = h(r')$, class $\overline{r}$
  analogously class $\overline{s}$

- nested loop join over $\overline{r}$, $\overline{s}$
  if $h(r) = h(s)$

Problem: Generation of partitions for large relations

3.5 Double-Hash-Join:

h_1 : partition R, S into large partitions



with

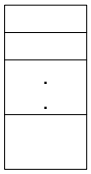
$|R_i|, |S_j| \leq AS$

cost: $|R| + |S|$ $O(n + m)$

condition: s may be join-partner of r only, if

$\exists i : r \in R_i \wedge s \in S_i$

h_2: read R_i in AS, hash with h_2

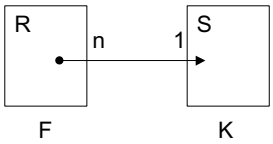


R_i is partitioned by h_2 into many small classes $R_{i,j}$

read S_i sequentially for $s \in S_i$ check partner in $R_{i,h2(s)}$
$O(n + m)$

3.6 Index-Join:

applicable for all n : 1 relationships



R.F = S.K

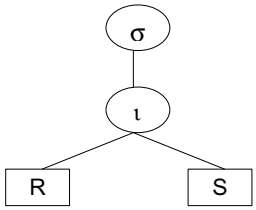
Evaluation: given r find s with r.F = s.K

r.F is key for s, find s via primary key

$O(|R| * \log |S|)$

3.7 Restrictions and Joins:

1. via primary key: $\sigma_P^1 \Rightarrow$ set of tuples
2. via secondary key: $\sigma_P^2 \Rightarrow$ TID-set
3. via relationsscan: $\sigma_P \Rightarrow$ set of tuples



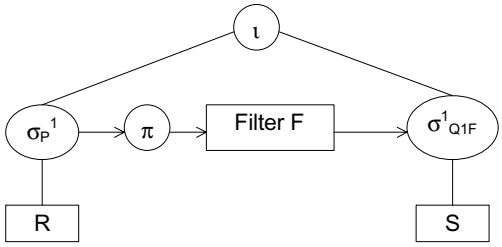
σ_P^1 R : Tuple-set for R and via foreign keys
TID-set for S = TID_S

σ_P^2 R : TID-set for R

σ_P R : tuple-set for R and via foreign keys
TID-set for S

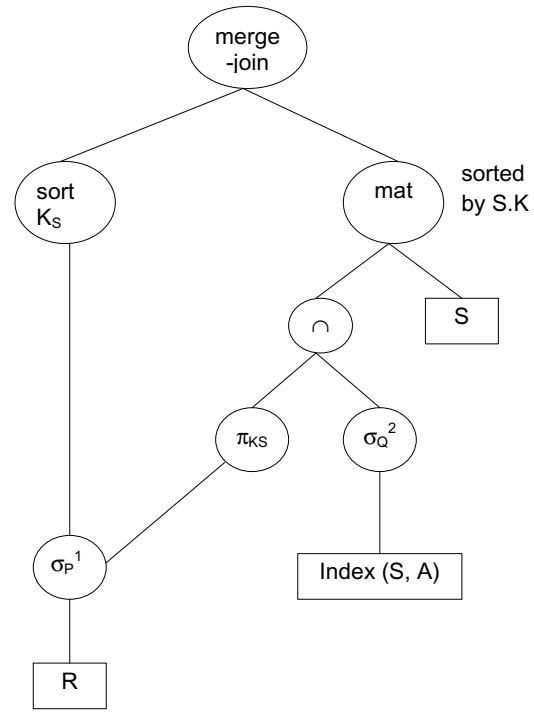
σ_P R with P ≡ **true** ~ relationscan

2. case: 2 primary indexes



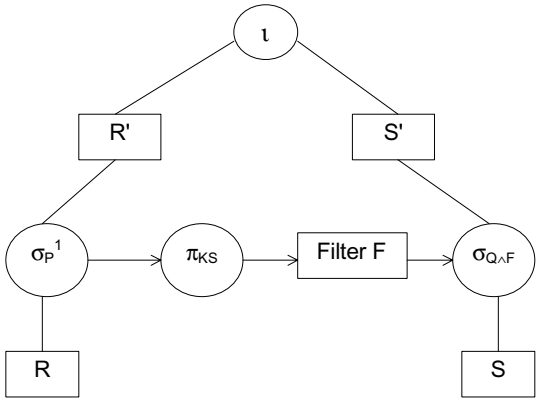
F ~ S.K ∈ TID_S

2. case: σ_P^1 for R
σ_Q^2 for S



with key K_R

case 3: σ_P^1 for R
σ_Q^2 for S



Note: R' is sorted by K_R
S' is sorted by K_S

⇒ sort the smaller of the 2 results for merge-join

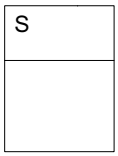
Exercise: 6 additional cases
duplicates?
costs?

4. Division: ÷

S ÷ T with (R x T) ÷ T = R

- T sort
- S sort according to (..., T-Attr.)

$S \div T = \{r \mid \forall t \in T : rt \in S\}$



cost:

$|S| * \log |S| + |T| * \log |T| + |S| + |T|$