

Lecture

# Database Implementation Techniques / Databases II

*OvGU Magdeburg, WinSem 2009/2010*

Sandro Schulze, Gunter Saake

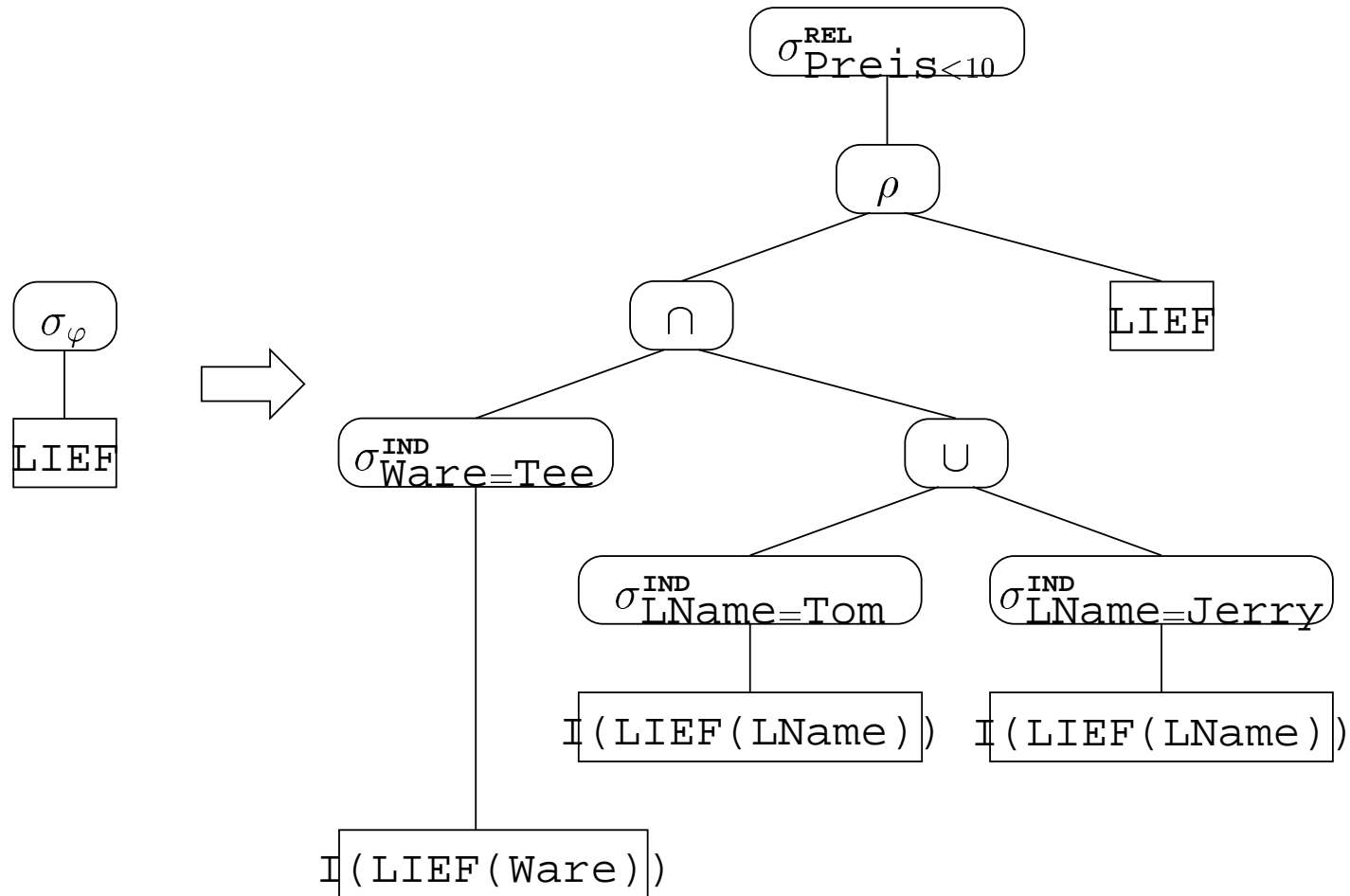
{sanschul,saake}@iti.cs.uni-magdeburg.de.

# Examples for internal Access Plans

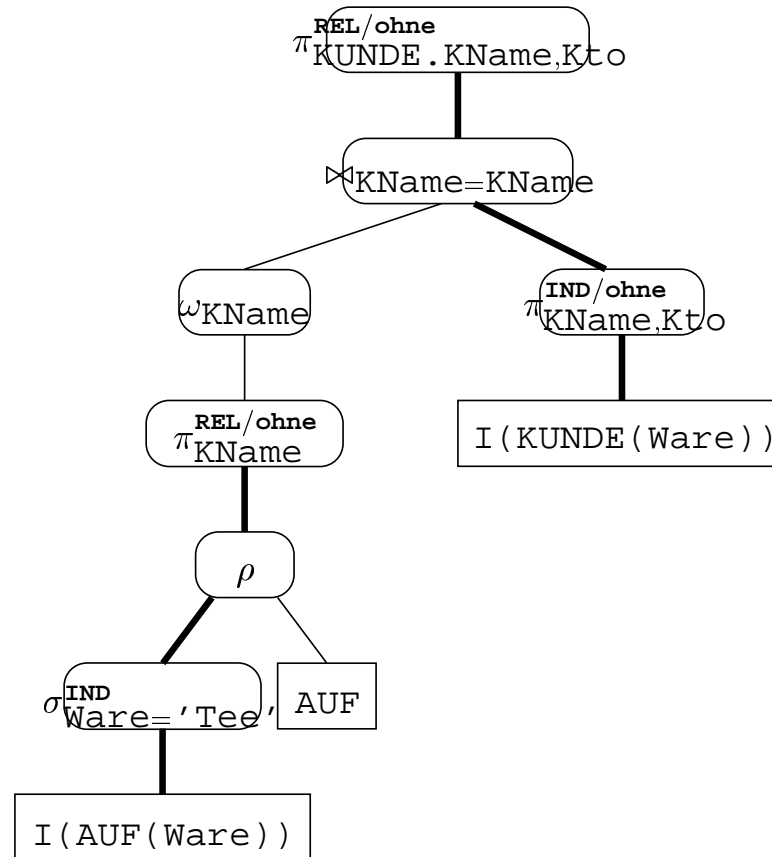
---

```
select *  
from DELIVERER  
where Product = 'Tea' and  
      ( DName = 'Tom' or DName = 'Jerry' )  
      and Price < 10
```

# Two Access Plans



# Pipelining of Operations



$$\pi_A(\sigma_\varphi(r(R))) \longleftrightarrow \langle \pi_A \circ \sigma_\varphi \rangle(r(R))$$

# Pipelining of Operations II

---

Typical combinations (fusions):

- Combination of Selection and Projection
- Combination of a Selection with Join
- Integration of a Selection in outer loop of Nested-Loop Join
- Integration of Selections in Merge Join
- Coupling of Selection with Realization operator

```
select C.CName, Kto  
from CUSTOMER C, ORDER O  
where C.CName = O.CName and O.Product = 'Tea'
```

# Shared Partial Queries

---

- Task: Detection of *shared partial queries*  $\rightsquigarrow$  the corresponding subtrees (of the operator tree) have to be put on the same level
- Problems:
  - ◆ Different syntactical form:  $r_1 \cup r_2$  identical with  $r_2 \cup r_1$
  - ◆ *Overlapping* ( $\sigma_\varphi$  overlaps  $\sigma_{\varphi \wedge \psi}$ )

# Cost-Based Selection

---

Considerations (for selection):

- Actual size of database relations
- Existence of indices (primary, secondary) and their size
- Clustering of several relations
- Selectivity of an attribute, an index has been defined on

# Relevant Database Parameter

---

- *System parameter* from catalogue:  $s$ : length of a page (useful page space in Byte)
- size of database:  $S$  as number of reserved pages (is needed, if tuples of relations are stored scattered)
- Statistical data about relations and indices:
  - ◆  $T_R$ : Number of tuples in relation  $R$
  - ◆  $L_R$ : Average length of a tuple in  $R$
  - ◆  $W_{A,R}$ : Number of different values for attribute  $A$  in  $R$  (index information or statistic)

Statistics → Update realized in case of change operation or by calling the respective command

# Selectivity of Attributes

---

Number of different values for attribute  $A$  in  $R$ :  $W_{A,R}$

- Equality:

$$sel(A=c, R) = \frac{1}{W_{A,R}}$$

- Inequality:

$$sel(\mathbf{not} A=c, R) = 1 - sel(A=c, R) = 1 - \frac{1}{W_{A,R}}$$

- Comparison using  $<$ ,  $>$ , ...:

$$sel(A < c, R) = sel(A > c) = \frac{1}{2}$$

# Selectivity of Attributes II

---

Refinement:

$$sel(A \leq c, R) = \frac{A_{max} - c}{A_{max} - A_{min}}$$

Range queries:

$$sel(c_u \leq A \leq c_o, R) = \frac{c_o - c_u}{A_{max} - A_{min}}$$

Selectivities for Joins:

$$sel_{\bowtie}(\varphi, R, S) \approx \frac{|R \bowtie_{\varphi} S|}{|R \times S|}$$

# Estimation of Selectivity

---

## 1. *Parametrized Functions*

Parameter of a function, which reflects well the data distribution, should be specified as exact as possible (e.g., Gaussian distribution)

## 2. *Histograms*

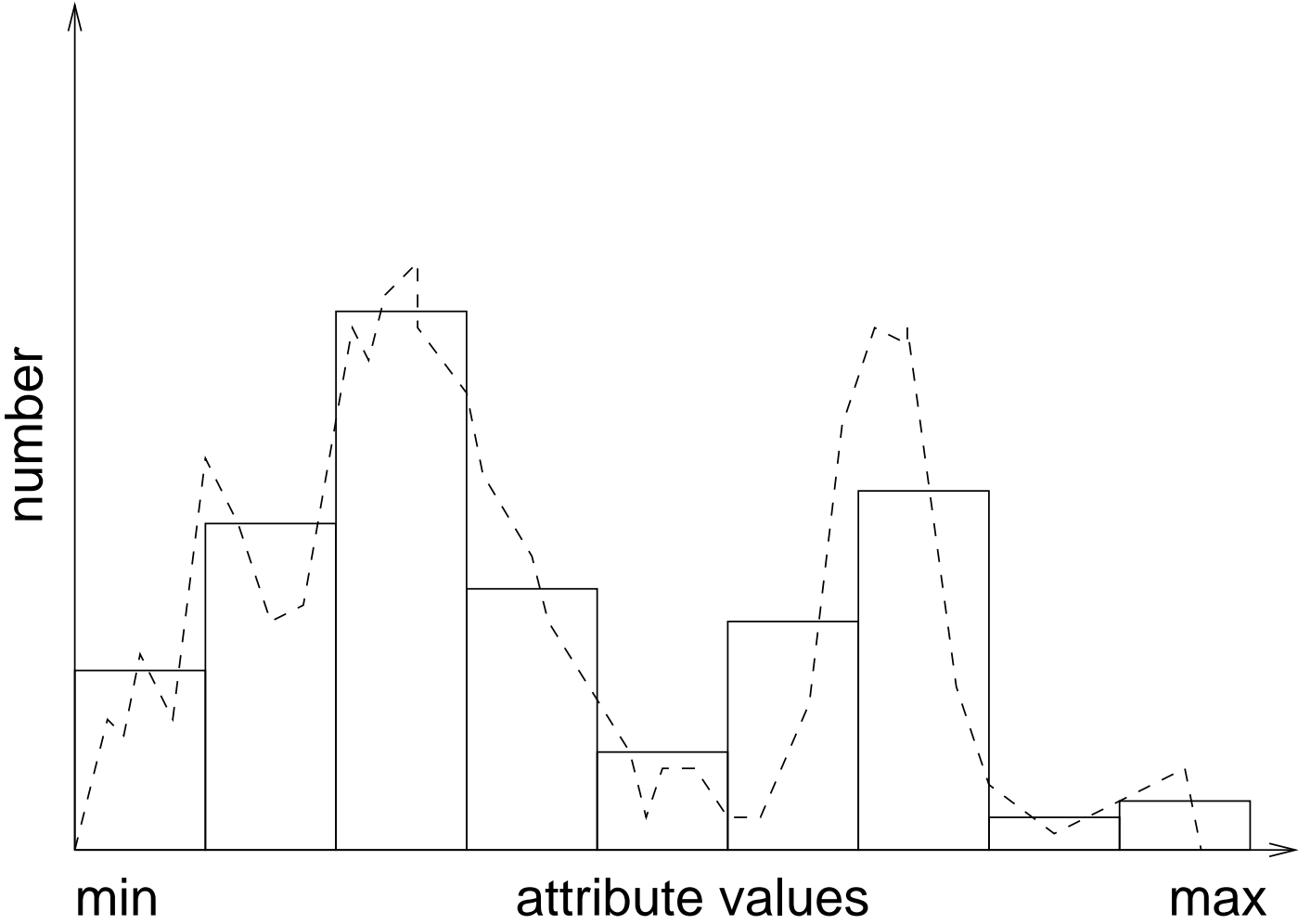
Domain is divided into subregions and the values, existing in reality (in these regions) have to be count

## 3. *Sampling*

Selectivity is determined with the help of random samples of the stored data records

# Histograms for Attribute Values

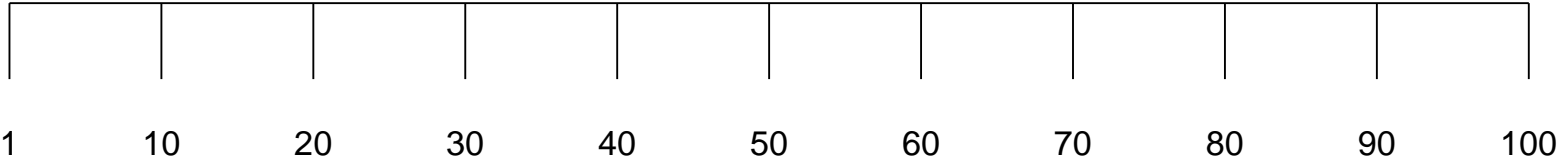
---



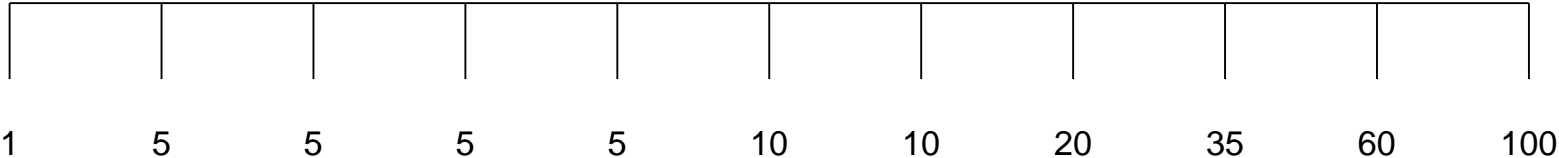
# Height-Balanced Histograms

---

a) uniformly distributed values



b) unequally distributed values



# Computation of Costs: Example

---

Selections on relation ORDER with attribute Product

- $S = 4.000$  (the database reserves 4000 pages overall)
- $T_{\text{ORDER}} = 10.000$  (in the relation ORDER 10.000 tuples are stored)
- $s/L_{\text{ORDER}} = 10$  (in average, 10 tuples fit into one page)
- $W_{\text{Product,ORDER}} = 50$ . (50 different products as value for the Product attribute)

Selection  $\sigma_{\varphi}$  mit  $\varphi = (A\theta a \wedge \psi)$

# Computation of Costs: Variant A

---

- Index  $I(R(A))$  on Selection attribute  $A$

$$\langle \sigma_{\psi}^{\mathbf{REL}} \circ \rho \circ \sigma_{A\theta a}^{\mathbf{IND}} \rangle (I(R(A)), r(R))$$

A1 Index with clustering:  $S_R \times sel(A\theta a, R)$ .

A2 Index without clustering:  $T_R \times sel(A\theta a, R)$ .

(Maximum value, if tuples are stored on different pages respectively)

# Computation of Costs: Variant B

---

- Index  $I(R(B))$  with  $B \neq A$

$$\langle \sigma_{A\theta a \wedge \psi}^{\mathbf{REL}} \circ \rho \circ \sigma_{\mathbf{true}}^{\mathbf{IND}} \rangle (I(R(B)), r(R))$$

B1 Index with clustering:  $S_R$ .

B2 Index without clustering:  $T_R$ .

Costs result from the fact, that the selectivity of the predicate **true** is 1

# Computation of Costs: Variant C

---

- Full table scan
- Assumption: All pages of the DB are read and all tuples, existing on these pages are found
- Costs: Given by the number  $S$

# Computation of Costs: Queries

---

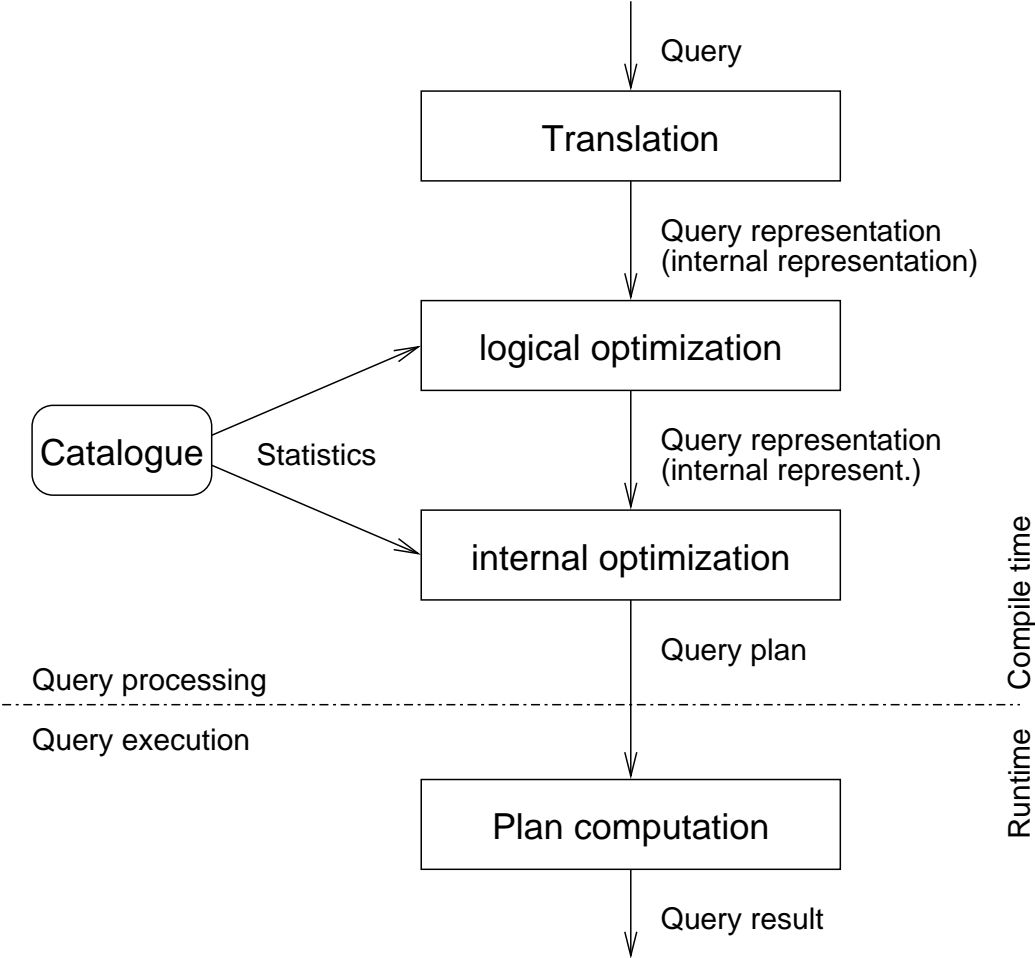
1.  $\sigma_{\text{Product}='Tea'}(r(\text{ORDER}))$ .  
Selectivity *sel*:  $\frac{1}{50}$
2.  $\sigma_{\text{Product}>'Tea'}(r(\text{ORDER}))$ .  
Selectivity *sel*: Assumption  $\rightsquigarrow \frac{1}{2}$ .

# Costs for Realization Vairants

---

<b>Variant</b>	<b>Cost formula</b>	Product = 'Tea'	Product > 'Tea'
A1	$S_R \times sel(A\theta a, R)$	20	500
A2	$T_R \times sel(A\theta a, R)$	200	5.000
B1	$S_R$	1.000	1.000
B2	$T_R$	10.000	10.000
C	$S$	4.000	4.000

# Optimizer Architecture



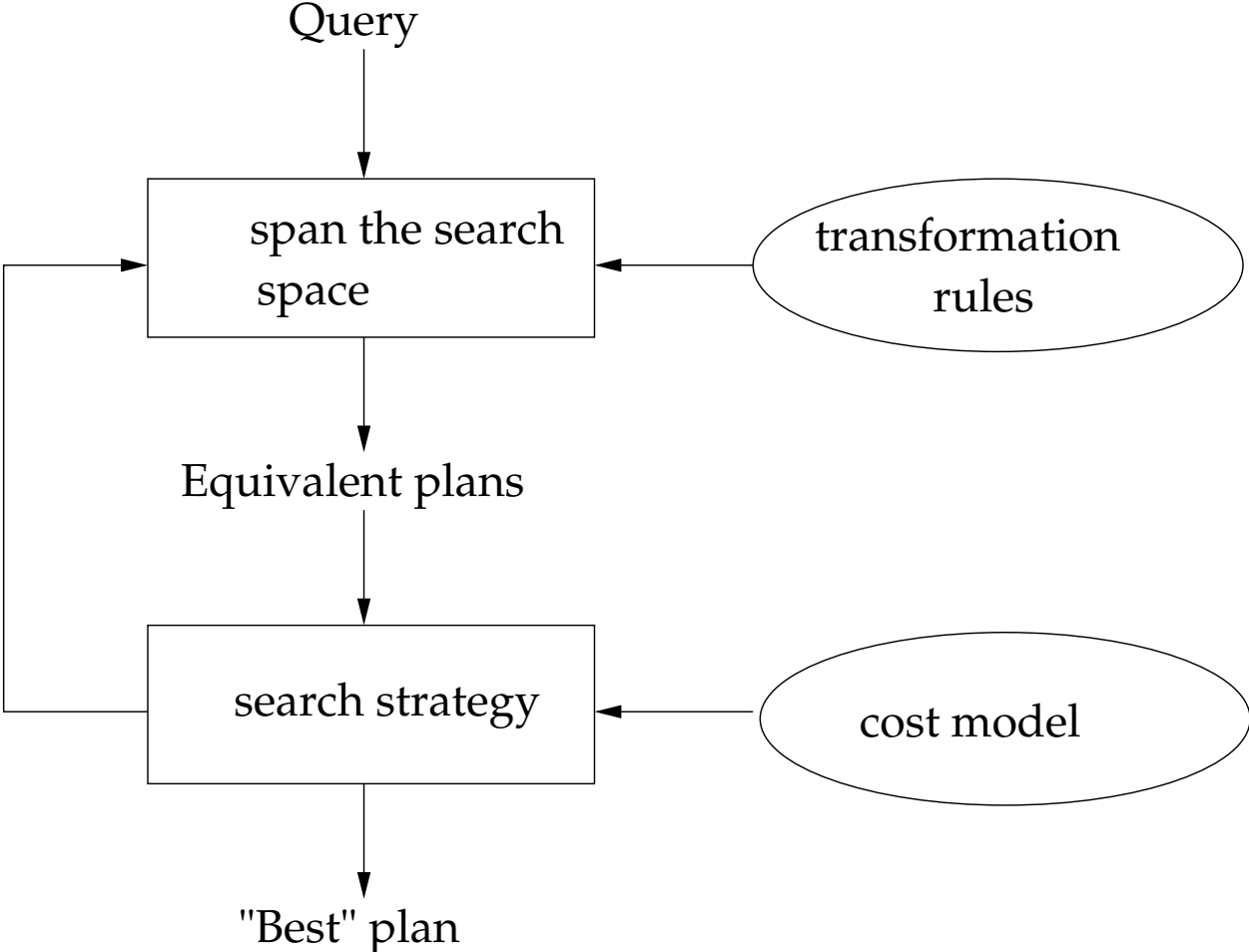
# Optimizer Variants

---

- *Heuristic, rule-based optimizer:*
  1. Generation of internal query representation by logical optimization
  2. Query plan is created by internal optimization
- *Cost-based (two-phase) optimizer:*
  1. Generation of different query representations by logical optimization
  2. Transfer to internal optimization (plan generation)
  3. Cost evaluation
  4. Selection of most suitable plan

# Optimization: Overview

---



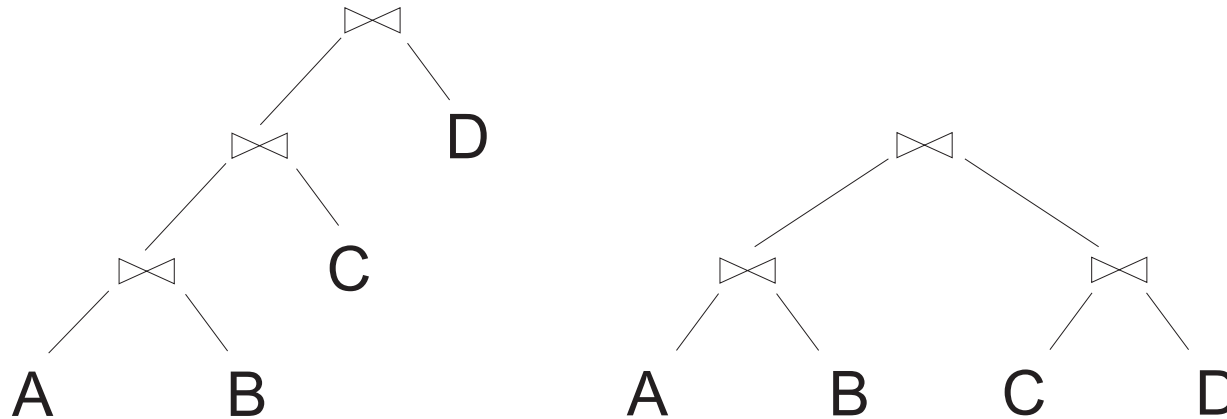
# Optimization: Search Space

---

- Search space: Set of all equivalent query plans
- Spanned by *transformation rules* (algebraic rules)
- Main focus: *Join Trees*
- For  $n$  relations:  $n!$  different Join trees!
- Hence: Limitation of search tree by
  - ◆ Heuristics (algebraic optimizations)
  - ◆ given „shape“ of the tree

# Optimization: Join Trees

---



- Linear sequence of operator trees
  - ◆ Only  $2^n$  Variants
  - ◆ *left deep tree* and *right deep tree*  $\rightsquigarrow$  all inner nodes of the tree possess at least one leaf node (basis relation) as child
- *bushy trees*
  - ◆ Higher potential for parallelization, though high effort for optimization

# Optimization: Search Strategies

---

- „Traversal“ of search space
- Selection of most cost-effective plan
- Foundation: Cost model
- Determines
  - ◆ *Which plans* are worth to be considered (complete / partial traversal)
  - ◆ The *order* in that alternatives are searched
- Variants: deterministic, random-based

# Optimization: Search Strategies /2

---

- deterministic:
  - ◆ Systematic generation of plans
  - ◆ Starts with plans for access on basis relations
  - ◆ Construction of complex plans by composition of more simple plans
  - ◆ Exhausting search; guarantees best plan
  - ◆ Example: *Dynamic Programming* (breadth-first search)
  - ◆ State of the art

# Optimization: Search Strategies /3

---

- Random-based:
  - ◆ One or more starting plans by Greedy strategy (depth-first search)
  - ◆ Improvement of starting plans by analyzing the „neighbors“
  - ◆ Neighbor: Application of transformation rules, e.g., exchange of two randomly selected operation
  - ◆ Example: *Simulated Annealing*
  - ◆ Better performance for huge number of relations
  - ◆ No guarantee for best plan

# Optimization in INGRES

---

- Dynamic technique for optimization
- Recursive partition of a calculus query in sequences of *Mono-Relation-Queries*
- Processing of mono-relation queries by „One Variable Query Processor“ (OVQP)
- OVQP selects best access method (Index selection)

# Optimization in INGRES /2

---

- Principle:

- ◆ Query  $q$ :

- select**  $R_2.A_2, R_3.A_3, \dots, R_n.A_n$   
**from**  $R_1, R_2, \dots, R_n$   
**where**  $P_1(R_1.A'_1)$  **and**  $P_2(R_1.A_1, R_2.A_2, \dots, R_n.A_n)$

- ◆ Partition in  $q'$ :

- select**  $R_1.A_1$  **into**  $R'_1$   
**from**  $R_1$   
**where**  $P_1(R_1.A'_1)$

- ◆ and  $q''$ :

- select**  $R_2.A_2, R_3.A_3, \dots, R_n.A_n$   
**from**  $R'_1, R_2, \dots, R_n$   
**where**  $P_2(R_1.A_1, R_2.A_2, \dots, R_n.A_n)$

# Optimization in INGRES /3

---

- Handling of unreducible multi-relation queries (esp. Join)
  - ◆ Converting to mono-relation queries by *tuple substitution*
  - ◆ For query  $q$ : Selection of a relation  $R_1$  and derivation of  $\text{card}(R_1)$  queries  $q'$  with  $n - 1$  relations

$q(R_1, R_2, \dots, R_n)$  replaced by

$$\{q'(t_{1i}, R_2, R_3, \dots, R_n); t_{1i} \in R_1\}$$

# Optimization in System R

---

- Dynamic Programming
- Bottom-Up construction of a plan
  1. Generation of simple plans (access on one relation)
  2. Generation of more complex plans (2 relations, 3 ... ) by combination (Join) of simple plans
- In the process: *Pruning*
  - ◆ Limitation of solution space by deletion of „unsound“ plans, equivalent (alternative) plans exist for
    1. Permutations with cartesian products
    2. Commutative strategies exhibit highest costs

# Optimization in System R /2

---

**Input:** SPJ query  $q$  on relations  $r_1, \dots, r_n$

**Output:** Query plan for  $q$

**for**  $i := 1$  **to**  $n$  **do**

$optPlan(\{r_i\}) := accessPlans(r_i)$

$prunePlans(optPlan(\{r_i\}))$

**end**

**for**  $i := 1$  **to**  $n$  **do**

**forall**  $s \subseteq \{r_1, \dots, r_n\}$  **such that**  $|s| = i$  **do**

$optPlan(s) := \emptyset$

**forall**  $t \subset s$  **do**

$optPlan(s) := optPlan(s) \cup$

$joinPlans(optPlan(t), optPlan(s - t))$

$prunePlans(optPlan(s))$

**end end end**

**return**  $optPlan(\{r_1, \dots, r_n\})$

# Optimization in System R /3

---

- Example query:

```
select E.EName
from Employees E, Assignment A, Project P
where E.ENr = A.ENo and A.PNr = P.PNr
and P.PName = 'DB development'
```

- **Indices:** Employees (ENr), Assignment (PNr), Project (PNr), Project (PName)

- Access plans after 1. iteration:

Employees: Full table scan

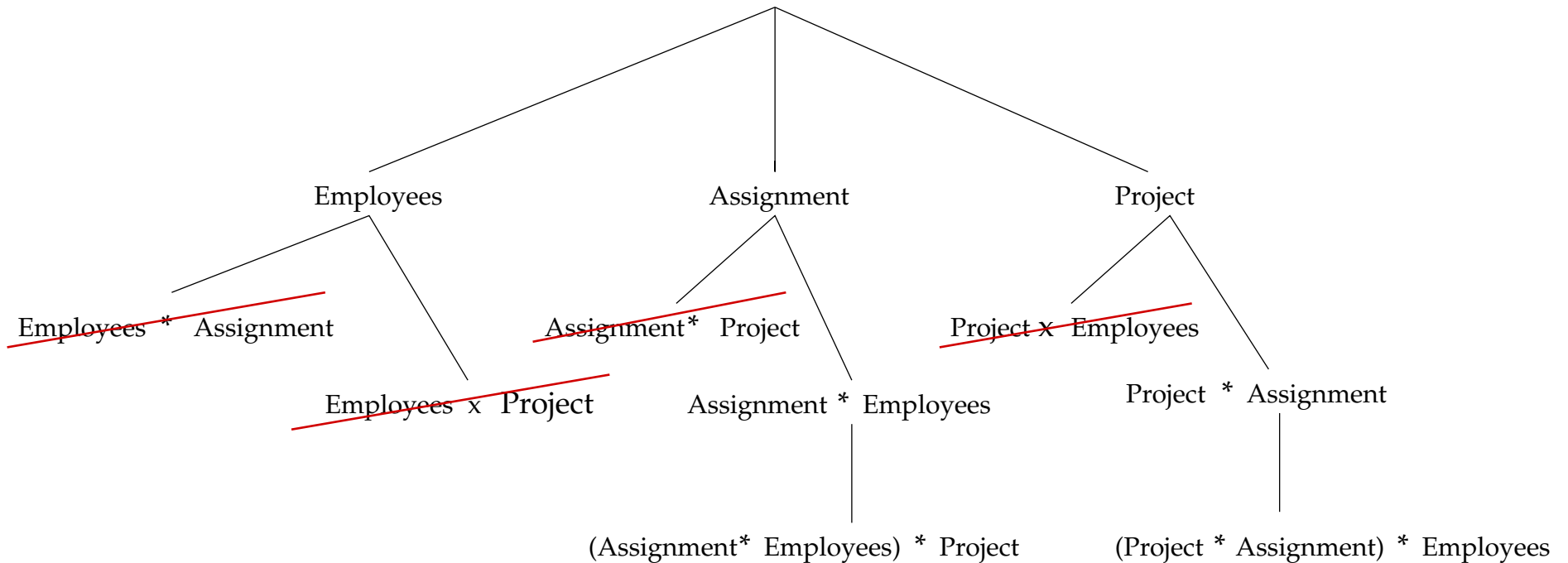
Assignment: Full table scan

Project: Index scan on PName

# Optimization in System R /4

---

## ■ Possible order of joins



# Oracle9i: Statistics

---

- Tables: Number of tables and blocks, averaged tuple length
- Columns: Number of different values, number of **NULL** values, data distribution (Histograms)
- Indices: Number of leaf pages, height of tree, clustering factor
- System: I/O- and CPU performance and workload respectively

# Retrieval of Statistics

---

- Retrieval by
  - ◆ Estimation based on samples (*row sampling, block sampling*)
  - ◆ Exact computation (Effort: Table scan + sorting)
  - ◆ user-defined
- Tools
  - ◆ **ANALYZE TABLE ...**
  - ◆ Package `DBMS_STATS`: Procedures for extended handling of statistics (recommended by oracle)

# Maintenance of Statistics

---

- Manual update of statistics (possibly as job)
- Automatic update (monitoring)
  - ◆ Observation of number of change operations
  - ◆  $\geq 10\%$  affected  $\rightsquigarrow$  out-dated data  $\rightsquigarrow$  update

# Missing Statistics

---

- Default values for missing statistics
  - ◆ Tables
    - Average size of tuples: 100 Bytes
    - Number of blocks: 1
    - Cardinality:  $num\_of\_blocks * (block\_size - cache\_layer) / avg\_row\_len$
  - ◆ Indices
    - Height: 1
    - Number of leaf pages: 25
    - Number of diff. key values: 100

# Generation of Statistics: ANALYZE

---

- Invocation:

```
analyze table table kind-of-statistic
```

- Kinds

- ◆ **estimate statistics**: Estimation by sampling; optional parameter specifies size of sample  
(**sample size rows | percent**

- ◆ **compute statistics**: Exact determination

- Example:

```
analyze table emp  
estimate statistics sample 10 percent;
```

# Generation of Statistics: DBMS\_STATS

---

- Invocation of package procedures:
  - ◆ Amongst others `gather_index_stats`,  
`gather_table_stats`, `gather_schema_stats`,  
...
- Example:

```
execute dbms_stats.gather_table_stats(  
    ownname => 'scott',  
    tabname => 'emp',  
    estimate_percent => NULL,  
    method_opt => NULL);
```

# DBMS\_STATS (II)

---

## ■ Parameter:

- ◆ `estimate_percent`: Estimation based on given sample size (in percent)
- ◆ `method_opt`: Specification of the statistics to be generated, u.a.
  - `FOR ALL [INDEXED ] COLUMNS`: all columns
  - `FOR COLUMNS column list`: for given columns
  - `AUTO`: automatic selection of columns for Histograms

# Display of Statistics

---

- Statistic data in Data Dictionary
- Views: dba\_-, user\_-, all\_tables, -tab\_col\_statistics
- Example: Table statistic (user\_tables)

TABLE_NAME	NUM_ROWS	BLOCKS	AVG_ROW_LEN
EMP	14	1	37

# Display of Statistics (II)

---

- Example: Column statistic  
(user\_tab\_col\_statistics)

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS
EMPNO	14	0	13
ENAME	14	0	13
JOB	5	0	4
MGR	6	1	5
HIREDATE	13	0	12
SAL	12	0	11
COMM	4	10	3
DEPTNO	3	0	2

# Oracle: Histograms

---

- Kinds
  - ◆ Height-based (equi-height)
  - ◆ Value-based (frequency)
    - Every value of column has corresponding bucket
    - Bucket number according to the frequency of the value
    - Applied, if number of different values of the column  $\leq$  number of buckets
- Selection depends on frequency of values

# Histograms: Generation

---

- **ANALYZE TABLE**

```
analyze table emp compute statistics  
for column sal size 10;
```

- **DBMS\_STATS**

```
execute dbms_stats.gather_table_stats(  
    ownname => 'scott',  
    tabname => 'emp',  
    method_opt => 'for columns size 10 sal');
```

- Default number of buckets: 75

# Output of Histograms

---

- Data Dictionary views: dba\_-, user\_-, all\_histograms

- Query:

```
select endpoint_number, endpoint_value
from user_histograms
where table_name='EMP' and column_name='SAL' ;
```

```
ENDPOINT_NUMBER  ENDPOINT_VALUE
-----
                1                800
                2                950
                3               1100
                ...
               13               3000
               14               5000
```

# Oracle: Output of Plans

---

- Storage of execution plan (incl. costs) for a certain query in a table `plan_table`

```
explain plan set statement_id = 'MPLAN'  
for select title from movie, budget  
where id between 2000 and 40000  
       and movie.id = budget.movie  
       and budget < 100000 and year = 1998;
```

# Oracle: Output of Plans (II)

---

- Readout of plan by query or specific tools

```
select substr(lpad(' ', 2*(level-1)), 1, 8) ||
       substr(operation, 1, 16) "OPERATION",
       substr(options, 1, 12) "OPTIONS",
       substr(object_name, 1, 12) object_name,
       id, parent_id, cost, cardinality, bytes
from plan_table
start with id=0 and statement_id = 'MPLAN'
connect by prior id = parent_id and
       statement_id = 'MPLAN';
```

# Oracle: Output of Plans (III)

---

## ■ Output

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID
-----	-----	-----	-----	----- . . .
SELECT STATEMENT			0	
NESTED LOOPS			1	0
TABLE ACCESS	BY INDEX ROW	MY_MOVIE	2	1
INDEX	RANGE SCAN	MOVIE_PK	3	2
TABLE ACCESS	BY INDEX ROW	BUDGET	4	1
INDEX	UNIQUE SCAN	SYS_C006658	5	4

# Oracle: Columns of Plan Table

---

Column	Meaning
statement_id	ID from <code>explain plan</code>
operations	plan operator (1. row: Statement)
options	details to plan operator
object_name	table, index etc.
id, parent_id	IDs of operations
position	(1. row: total costs, otherwise: relative pos.)
cost	costs of operation (function on CPU/IO costs)
cardinality	number of processed tuples
bytes	size of processed Bytes
cpu_cost, io_cost	CPU/IO costs (proportional to actual values)
temp_space	required temp. storage in Bytes

# Oracle: Operations of Plan table

---

operation	option	Meaning
filter		Selection wrt. condition
hash join		Hash Join
merge join		Merge Join
merge join	(outer, cartesian)	
nested loops		Nested-Loops Join
index	unique scan	Access to single Rowid
index	range scan	Range access on index
sort	group by	Sorting for grouping
sort	unique	... for duplicate elimination
sort	join	... for Merge Join
table access	full	table scan
table access	by index rowid	table access on Rowid of index access

# Oracle: Optimizer Hints

---

- Specific influence on optimization of queries
- Aspects:
  - ◆ Objective: Throughput / response time
  - ◆ Query transformation (e.g., star queries, materialized views etc.)
  - ◆ Access paths
  - ◆ Join order
  - ◆ Join operation
- Indication: by comments
  - /\* +hint \*/*
  - +hint*

# Hints: Throughput vs. Response Time

---

- Goal of optimization: highest throughput (minimal resource usage)

```
select /*+ all rows */ *  
from emp;
```

- Goal of optimization: best response time, i.e., first  $n$  tuples should be provided as fast as possible (not for **group by**, **order by**, set operations and **distinct** queries)

```
select /*+ first rows(10) */ *  
from emp;
```

# Hints: Access Paths

---

- Selection of different strategies: full, index, ...
- Example: Usage of index for ename

```
select /*+ index(emp ename_idx) */ *  
from emp  
where ename = 'JONES' ;
```

# Hints: Joins

---

- Specification of Join implementation

- ◆ `/* +use_nl(inner_tbl) */`: Nested-Loops Join

- ◆ `/* +use_merge(tbl1 tbl2) */`: Merge Join

- ◆ `/* +use_hash(tbl1 tbl2) */`: Hash Join

- Specification of Join order

- ◆ `/* +ordered */`: Order like in **from** clause

- Example:

```
select /*+ ordered */ *  
from tab1, tab2, tab3  
where tab1.col = tab2.col  
and tab1.col = tab3.col;
```