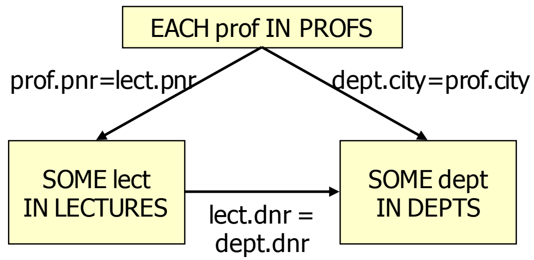
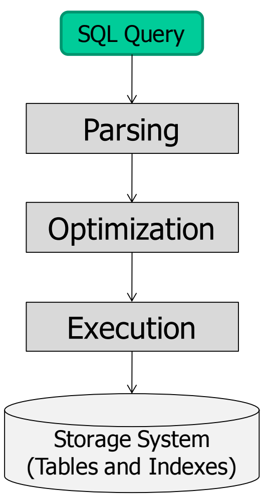
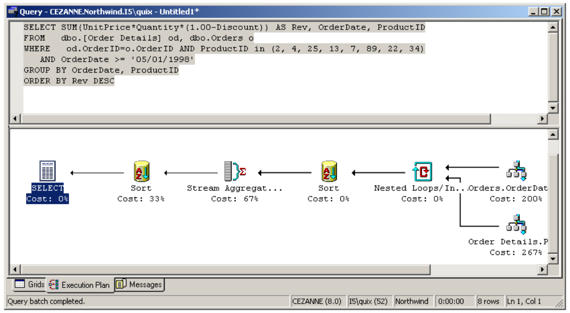
Chapter 3
Query Optimization

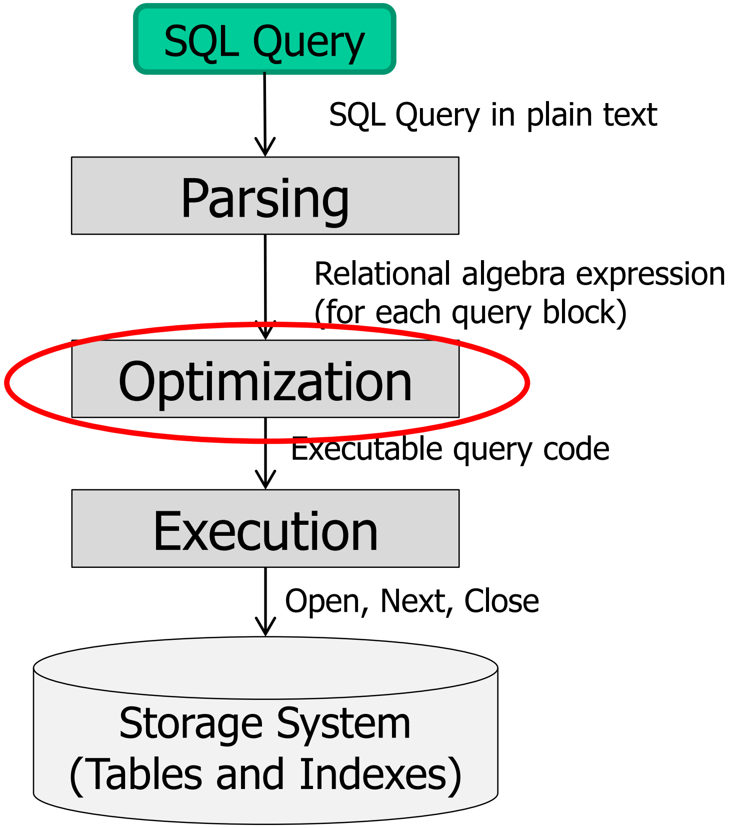




eno	name	marstat	salary	dno	dname	mgr	
	a2						
b1	a2	single	b2	b3			c EMPL
				b3	computer	b4	d DEPT
b1	b5	single	<40.000	b6			t EMPL



Contents



SQL Query
SQL Query in plain text
Parsing
Relational algebra expression
(for each query block)
Optimization
Executable query code
Execution
Open, Next, Close
Storage System
(Tables and Indexes)

3.1 Semantic Query Optimization - Advantages

- **Faster evaluation as less joins are used for**
  - Relational queries
  - Deductive queries on fact/rule systems
  - Natural language queries (term definitions correspond to rules)

- **Explanation of unexpected results**
  - Violation of integrity constraints (leads to empty results)
  - Trivial queries (leads often to very large results)
  - Pre-supposition

$\Rightarrow$ **Answer:** Rules instead of fact lists

Motivating Example – Semantic Optimization of Views (1)

Names of single computer people in a tax bracket under 50%

Query:

```
SELECT c.name FROM COMPEMP c, TAX50 t

WHERE  c.marstat=´single´ AND t.eno=c.eno
```

Views:

```
CREATE VIEW COMPEMP AS
    SELECT e.* FROM EMPL e, DEPT d
    WHERE d.dname='computer' AND d.dno=e.dno;
```

```
CREATE VIEW TAX50 AS
    SELECT e.* FROM EMPL e
    WHERE  (e.marstat='single' AND e.salary<40.000)
    OR   (e.marstat='married' AND e.salary<80.000);
```

Motivating Example – Semantic Optimization of Views (2)

View Substitution

```
SELECT c.name FROM EMPL c, DEPT d, EMPL t
WHERE  d.dname='computer' AND c.dno=d.dno AND
       c.marstat='single' AND
       t.marstat='single' AND
       t.salary<40.000 AND c.eno=t.eno

OR     d.dname='computer' AND c.dno=d.dno AND
       c.marstat='single' AND
       t.marstat='married' AND
       t.salary<80.000 AND c.eno=t.eno
```

→ Tableau method for simplification!

Tableau Representation

[Aho et al., 1979; Abiteboul et al., 1995]

- Representation for a special class of *conjunctive queries* in domain relational calculus (DRC):

  $$\{a_1 \ldots a_m \mid \exists\, b_1 \ldots b_n\ (P_1 \wedge P_2 \wedge \ldots \wedge P_k)\},$$

  where P_i are atomic predicates (relation predicates or comparisons).

- For each relation predicate the tableau contains a row, and for each variable a column.

- **Restriction:** Attributes are expected to appear only once per relation (*unique role assumption*).

- Tableaux correspond to SELECT-PROJECT-JOIN (**SPJ**) queries in RA

  → construction of tableaux by translating RA expressions into equivalent DRC queries

Tableau Method – Example (1)

```
SELECT c.name FROM EMPL c, DEPT d, EMPL t

WHERE  d.dname='computer' AND c.dno=d.dno AND
       c.marstat='single' AND t.marstat='single' AND
       t.salary<40.000 AND c.eno=t.eno

OR     d.dname='computer' AND c.dno=d.dno AND
       c.marstat='single' AND
       t.marstat='married' AND
       t.salary<80.000 AND c.eno=t.eno
```

Equivalent query in Domain Relational Calculus (DRC):

{ n | ∃dname, dno, mst, sal, eno, mgr, n2, mst2, sal2, dno2

EMPL(eno,n,mst,sal,dno) ∧ DEPT(dno,dname, mgr) ∧

EMPL(eno,n2,mst2,sal2,dno2) ∧

dname='computer' ∧ mst='single' ∧ mst2='single' ∧ sal2<40.000 } ∪

{ n | ∃ dname, dno, mst, sal, eno, mgr, n2, mst, sal2, dno2

EMPL(eno,n,mst,sal,dno) ∧ DEPT(dno,dname, mgr) ∧

EMPL(eno,n2,mst2,sal2,dno2) ∧

dname='computer' ∧ mst='single' ∧ mst2='married' ∧ sal2<80.000}

Tableau Method – Example (2)

{ n | ∃ dname, dno, mst, sal, eno, mgr, n2, mst, sal2, dno2
EMPL(eno,n,mst,sal,dno) ∧
DEPT(dno,dname, mgr) ∧
EMPL(eno,n2,mst2,sal2,dno2) ∧
dname='computer' ∧ mst='single' ∧ mst2='single' ∧ sal2<40.000 }

eno	name	marstat	salary	dno	dname	mgr	
	a2						
b1	a2	single	b2	b3			EMPL
				b3	computer	b4	DEPT
b1	b5	single	<40.000	b6			EMPL

Syntactic simplification: Removal of superseded rows

Tableau Method – Example (3)

$\{\ n \mid \exists$ dname, dno, mst, sal, eno, mgr, n2, mst, sal2, dno2

EMPL(eno,n,mst,sal,dno) $\wedge$

DEPT(dno,dname, mgr) $\wedge$

EMPL(eno,n2,mst2,sal2,dno2) $\wedge$

dname='computer' $\wedge$ mst='single' $\wedge$ mst2='married' $\wedge$ sal2<80.000}

eno	name	marstat	salary	dno	dname	mgr	
	a2						
b1	a2	single	b2	b3			EMPL
				b3	computer	b4	DEPT
b1	b5	married	<80.000	b6			EMPL

Semantic constraint propagation ("chase") and deletion of contradictory tableaux.

Tableau Method – Example (4)

Result Tableau

eno	name	marstat	salary	dno	dname	mgr	
	a2						
b1	a2	'single'	<40.000	b3			EMPL
				b3	'computer'	b4	DEPT

{ n | ∃ dname, dno, mst, sal, eno, mgr

EMPL(eno,n,mst,sal,dno) ∧

DEPT(dno,dname, mgr) ∧

dname='computer' ∧ mst='single' ∧ sal<40.000 }

```
SELECT c.name FROM EMPL c, DEPT d

WHERE  d.dname='computer' AND c.dno=d.dno AND
       c.marstat='single' AND
       c.salary<40.000
```

Tableau Containment and Equivalence (1)

- **Optimization Goal:**
  Find the minimal tableau of all equivalent tableaux.

- Reduction of tabular rows results in reduction of the number of necessary (expensive) joins.

Definition 3.1

Tableau T_1 is *contained* in tableau T_2 ($T_1 \subseteq T_2$) if

1. T_1, T_2 have the same columns and entries in result rows **and**
2. the relation computed from T_1 is a subset of the one from T_2
   for all valid assignments of relations to rows and
   for all valid database instances.

- Tableau Containment $\Rightarrow$ Query Containment

Tableau Containment and Equivalence (2)

Theorem 3.1 (Homomorphism Theorem [Abiteboul et al., 1995])

$T_1 \subseteq T_2 \Leftrightarrow$
There is a mapping h from the T_2 symbols to the T_1 symbols with:

1. h(resulting_row(T_2))= resulting_row(T_1)
2. h(row(T_2))= any row of T_1 with the same relation name
3. h(constant) = constant
4. Integrity constraints in T_2 are transferred to the respective symbols in T_1 and are also guarenteed in T_1.

Theorem 3.2

Two tableaux T_x and T_y are equivalent, denoted as ($T_x \equiv T_y$)
$\Leftrightarrow T_x \subseteq T_y \wedge T_y \subseteq T_x$

Example Tableau Containment

$T_1 \subseteq T_2$? → Find mapping h from T_2 to T_1
$T_2 \subseteq T_1$? → Find mapping h from T_1 to T_2

T_1

a		
a	b	(R)
c	d	(R)
e	f	(R)

b<d, d<f

T_2

w		
w	x	(R)
y	z	(R)

x<z

Example Tableau Containment

$T_3 \subseteq T_4$? → Find mapping h from T_4 to T_3
$T_4 \subseteq T_3$? → Find mapping h from T_3 to T_4

T_3

a		b	
a	c	d	(R)
a	e	f	(R)
g	c	b	(R)
h	e	b	(R)

T_4

a		b	
a	e	f	(R)
h	e	b	(R)

Tableau Minimization

- For each tableau row:
  Delete the row and check equivalence to original tableau

- Unfortunately, this minimization is NP-complete (no problem for small tableaux)

> **Theorem 3.3**
> Every *minimal tableau* is equivalent to the found tableau (except naming).

- Apply **integrity constraints** ("knowledge-based optimization" using special reasoners, e.g. *chase algorithm*) to find minimal equivalent tableau
- **Key constraints / functional dependencies**: If left sides of FDs (keys) are equal in 2 tableau rows, then right sides are equal as well.
- **Referential constraints**: "pending" rows can be eliminated.
- **Domain constraints**: Constant propagation and elimination of unnecessary comparisons

3.2 Structure-based Query Optimization

- Based on representation of queries as graphs
- Analysis and illustration of structural query properties
- Description of special techniques for query evaluation
- There are various types of query graphs differing in expressiveness:
  - Syntax trees
  - **Quant graphs** [Jarke & Koch, 1983; Jarke & Koch, 1984]

Classification of Queries

- Classification by number of variables (in TRC)
  - **One variable expressions** (one dimensional indexes support the evaluation of single or conjunct connected *monadic terms*)
  - **Two variables expressions** (*dyadic terms*, can be reduced to components with 2 variables,)
  - **Multiple variables expressions**
- Classification by junctors and quantifiers
  - Special case: conjunctive queries
- Classification by structure of the graphical representation of the query
  - **Tree-like queries**
  - **Cyclic queries**

Syntactical Preliminaries

The *scope* of a variable (function SC(v)) is the subexpression of the query, in which a variable may be used.

Example:

Scope of p

$\{e \mid e \in EMPL \wedge e.name = \text{'Müller'} \wedge$

$(\neg(\exists p \in PAPERS\ \boxed{(p.year=2011 \wedge p.eno=e.eno))} \vee$

$\exists c \in COURSES \wedge \exists t \in TIMETABLE$

$(c.level = \text{'Bachelor'} \wedge (t.cno=c.cno \wedge$

$t.eno=e.eno)))\}$

We use $\{ r \in rel \mid \Phi \}$ as shortcut for $\{ r \mid r \in rel \wedge \Phi \}$

Quant Graphs

Given an expression in TRC with conjunctively connected dyadic terms over free, existentially or universally quantified variables:

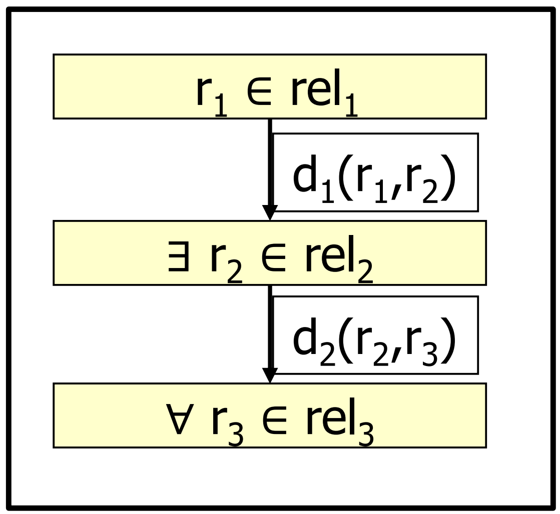
Blank node for free result variables

- *Range terms* "$quant_i\ r_i \in rel_i$" with $quant_i \in \{\exists, \forall, \_\}$, $rel_i \neq \emptyset$, are represented as nodes k_i.
- *Dyadic comparison terms* $d(r_i, r_j)$ are represented as directed edges $k_i \rightarrow k_j$ and labeled with $d(r_i, r_j)$. Is d "=", label of the edge is "=".
- The *direction* of edge $k_i \rightarrow k_j$ indicates: $SC(r_j) \subseteq SC(r_i)$.
- A relation $SC(r_j) \subseteq SC(r_i)$ that is not the result of a dyadic predicate is represented as an *unlabeled edge* $k_i \rightarrow k_j$.

Quant Graphs – First Example

$$\{ r_1 \in rel_1 \mid \exists\, r_2 \in rel_2\ \forall\, r_3 \in rel_3\ (d_1(r_1,r_2) \wedge d_2(r_2,r_3))\}$$

$$\equiv\{r_1 \in rel_1 \mid \exists\, r_2 \in rel_2\ (d_1(r_1,r_2) \wedge \forall\, r_3 \in rel_3\ (d_2(r_2,r_3))\}$$



- Expressions are equivalent if range of values of variable r_3 is not empty
- Definition ($rel_i \neq \emptyset$) then guarantees that only equivalent expressions can be described by the same Quant Graph

Structural Query Properties (1)

- A *path* between two nodes k_i and k_j in the Quant Graph is a sequence of together stringed edges connecting the both nodes.

- If all path edges are labeled, the path is called *predicate path*.

- If all pairs of adjacent edges have the same direction, then the path is *directed* (otherwise *undirected*).

- A *cycle* (*predicate cycle)* is an undirected path (predicate path) connecting a node with itself.

Structural Query Properties (2)

- A Quant Graph is called *strongly connected*, if for all nodes k_i there is an undirected predicate path to every other node k_j of the graph.

- A strongly connected Quant Graph without cycles is called *strictly tree-like*.

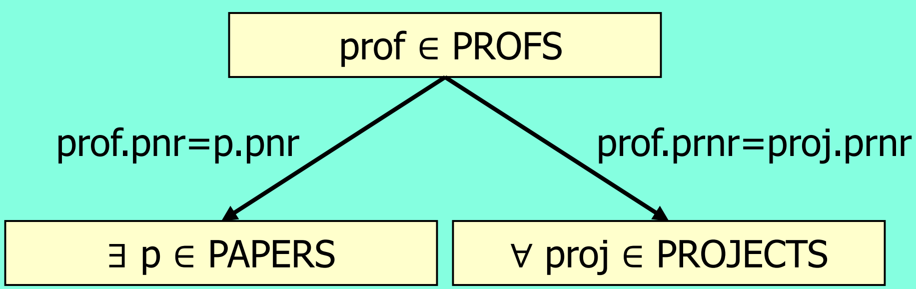
- A strongly connected Quant Graph without predicate cycles is called *simply tree-like*.

- A Quant Graph with at least one predicate cycle is called *cyclic*.

Quant Graphs – Further Examples

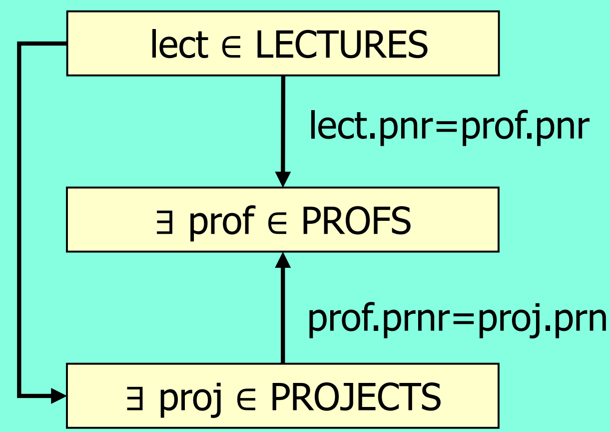
a) Strictly tree-like Quant Graph:

$$\{\ prof \in PROFS \mid \exists\, p \in PAPERS\ (prof.pnr = p.pnr) \land \forall\, proj \in PROJECTS\ (prof.prnr = proj.prnr)\ \}$$



b) Simply tree-like Quant Graph:

$$\{\ lect \in LECTURES \mid \exists\, proj \in PROJECTS\ \exists\, prof \in PROFS\ (lect.pnr = prof.pnr \land prof.prnr = proj.prnr)\}$$

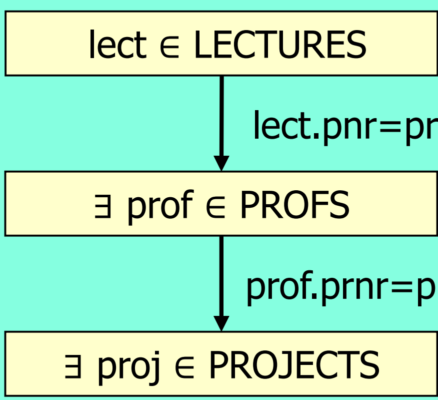


Examples of Quant Graphs

c) Exchange of quantifiers of the same type in b) leads to an equivalent expression

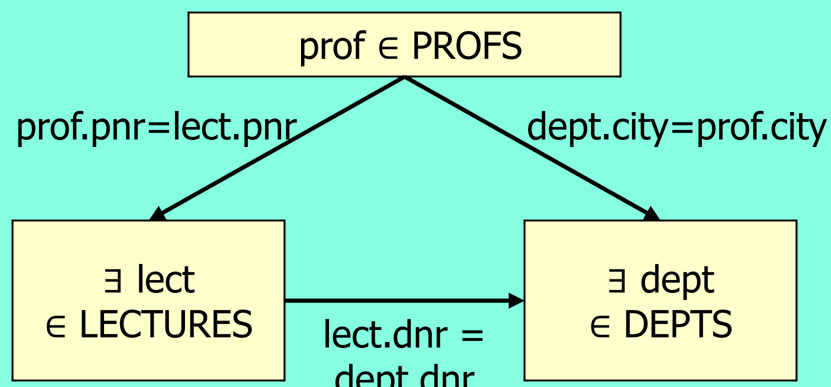
$\{$ lect $\in$ LECTURES $|$
$\exists$ prof $\in$ PROFS
$\exists$ proj $\in$ PROJECTS
(lect.pnr=prof.pnr $\wedge$
prof.prnr=proj.prnr)$\}$

with strictly tree-like Quant Graph:



d) Cyclic Quant Graph:

$\{$ prof $\in$ PROFS $|$
$\exists$ lect $\in$ LECTURES
$\exists$ dept $\in$ DEPTS
(prof.pnr=lect.pnr $\wedge$
lect.dnr=dept.dnr $\wedge$
dept.city=prof.city)$\}$



Problem with Cyclic Quant Graphs

- There is no sequence of semijoin operators that compute the correct result for arbitrary database states.
- There are DB states where no possible sequence of operands can achieve a reduction of the involved range relations.
- Sample DB state:

PROFS

pnr	pname	status	city
1	Bolour	assistent	Berkeley
2	Wasserman	tenure	San Francisco

LECTURES

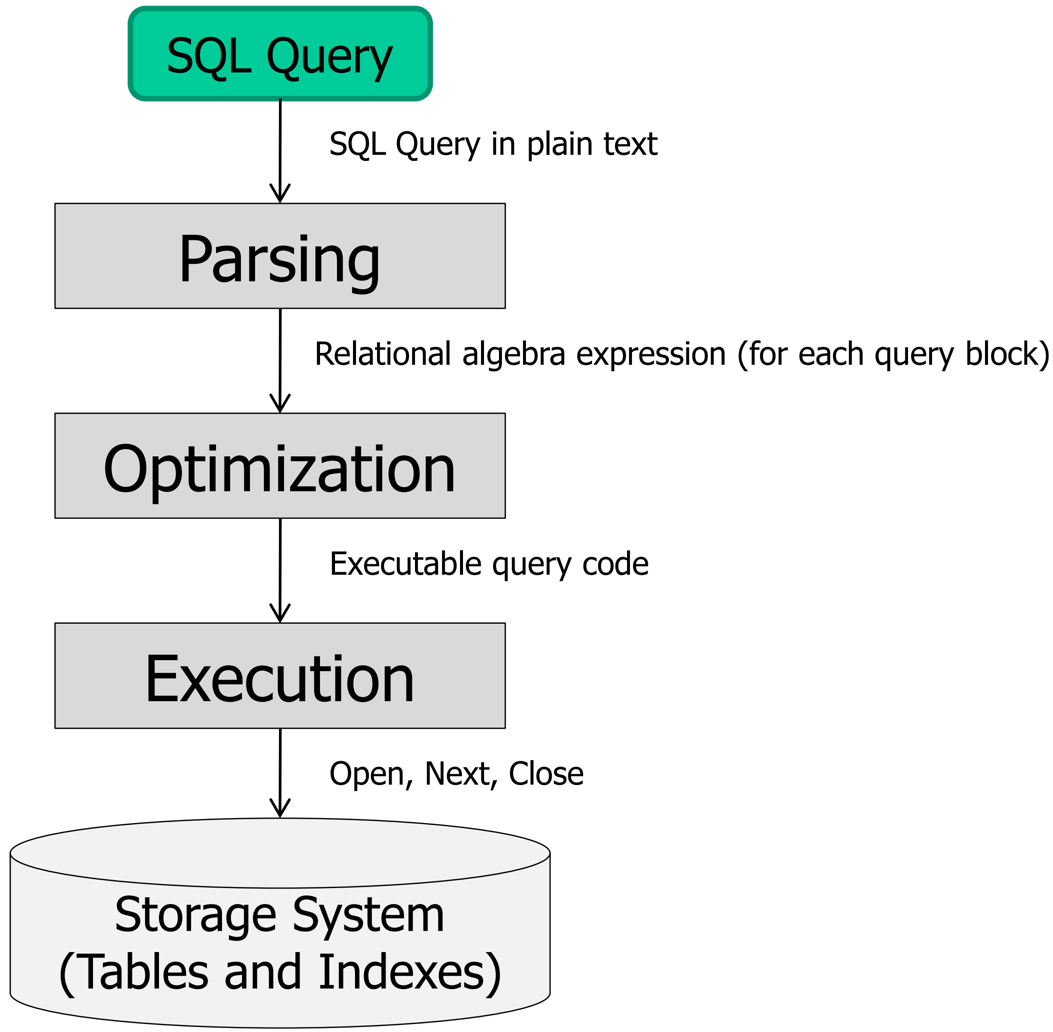
dnr	pnr	room	day	daytime
47	1	502	Tuesday	8:00
20	2	603	Friday	10:00

DEPTS

dnr	dtype	city
47	medicine	San Francisco
20	computer science	Berkeley

- Each operation $rel_1 \ltimes rel_2$ with rel_1, $rel_2 \in$ {PROFS, LECTURES, DEPTS} creates for the given DB state the (unrestricted) relation rel1 as intermediate result.

3.3 Cost-based Query Optimization



Optimization of a Relational Query

- Semantic Query Optimization
- Structure-based Query Optimization
- **Cost-based Query Optimization**
  - Retrieve statistical information about relations (size, access paths)
  - Access planning
    - Block sequence
    - Planning in each block:
      - Join sequence
      - Join methods
      - Access paths

Relational Algebra Equivalences

- **Selection:**
  - $\sigma_{c1 \wedge \ldots \wedge cn}(R) = \sigma_{c1}(\ldots(\sigma_{cn}(R)\ldots)$
  - $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
- **Projection:**
  - $\Pi_{a1}(R) = \Pi_{a1}(\ldots(\Pi_{an}(R) \ldots)$
- **Join:**
  - $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$
  - $R \bowtie S = S \bowtie R$

Relational Algebra Equivalences (2)

- A projection commutes with a selection that only uses attributes retained by the projection.

- Selection between attributes of the two arguments of a cross-product converts cross-product to a join:

$$\sigma_{R.X=S.Y}(R \times S) = R \bowtie_{R.X=S.Y} S$$

- A selection just on attributes of R commutes with $R \bowtie S$

$$\sigma_c(R \bowtie S) = \sigma_c(R) \bowtie S$$

- Similarly, if a projection follows a join $R \bowtie S$, we can `push' it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.

Query Optimization
(Selinger-style / System-R-Style)

- **Many ways to execute SQL Query**
  - Algebraic properties
  - Choice of operator implementations
  - Costs may be very different (see comparison of join implementations)

- **Algebraic Transformations**
  - Equivalence Transformations
  - Problem: Exponential number of equivalent expressions (e.g., n! for n joins)

- **Estimation Model (Cost Model)**
  - Needs to estimate the costs of an operator (or a sequence of operators) and the size of its result

- Query Optimization in System R (IBM 1979), still the foundation for QO in DBMS today [Selinger et al., 1979]

Cost Estimation for a Single elation (only one access path is searched)

- Basic cost formula:

Secondary storage accesses (page fetches)

$$COST = PF + w \cdot (predicted\_\#tuples)$$

Weighting factor between I/O and CPU time

- Statistical data:
  - For each relation R:
    - B(R): number of pages needed to store relation R
    - T(R): number of tuples of relation R
  - For each attribute a of R:
    - V(R,a): number of distinct values relation R has in attribute a

- Selectivity factors F:
  Selectivity factors estimate the effect of a restriction

[Selinger et al., 1979, Garcia-Molina et al., 2009]

Estimation of Selectivity Factors

Condition	Selectivity Factor F	Remarks
col=value	1/V(R,col)	1/10 if there is no index
col1=col2	1/max(V(R,col1),V(R,col2))	1/10 if there is no index
col>val	(highkey - val) / (highkey - lowkey)	1/3 if col. is not arithmetic
col BETWEEN val1 AND val2	(val2-val1) / (highkey - lowkey)	1/4 if col. is not arithmetic
col IN (list OF val)	min(|list| · F(col=value), ½)	
col IN subquery	(estimated cardinality of subquery result) / ($\Pi_{R_i \text{ IN FROM-List of subquery}}$ T(R_i))	
pred1 OR pred2	F(pred1) + F(pred2) – F(pred1) · F(pred2)	
pred1 AND pred2	F(pred1) · F(pred2)	Assumption: Column values must be indepdt.
NOT pred:	1- F(pred)	

Note: There might be multiple ways to compute the selectivity for a predicate! [Selinger et al., 1979]

Dynamic Programming

- **Goal:** Find the optimal plan for Join($R_1,\ldots,R_n$)
  - For each S in $\{R_1,\ldots,R_n\}$ do:
    Find optimal plan for Join(Join($\{R_1,\ldots,R_n\} \setminus S$), S)
  - Pick the plan with the lowest total cost

- **Principle of Optimality:**
  - Optimal plan for a larger expression requires optimal plans for its sub-expressions

- **Complexity**
  - Enumeration cost drops from $O(n!)$ to $O(n \cdot 2^n)$
  - May need to store $O(2^n)$ partial plans
  - Significantly more efficient than the naïve scheme

Single-relation Queries

Choosing cost minimal access path for single-relation queries

- Assumption about buffer management for clustered and non-clustered indexes
- "interesting order" specified in query by ORDER BY and GROUP BY $\Rightarrow$ only interesting:
  - All paths with the "right" order
  - The cheapest plan with any other order
    - Orders which might be beneficial for a subsequent merge join
- If no "interesting order" is given, only cheapest plan is of interest

Cost Formulas for Single-relation Queries

Context	**Cost**
Unique index for equal predicate	$\log_G 0.15B(R)+w$ Note: Only one result tuple!
Applicable* clustered index	$\log_G 0.15B(R)+ F(pred) \cdot B(R) + w \cdot \#tuples$
Applicable non-clustered index	$\log_G 0.15B(R)+ F(pred) \cdot T(R)+ w \cdot \#tuples$
Scan of Relation	$B(R) + w \cdot T(R)$

* **Applicable:** Index I contains column of one or more conjunct in WHERE clause of conjunctive query

[Selinger et al., 1979]

See also slide 31 in chapter 2

Cost Estimation and Join Strategies

- Methods:
  - All applicable join methods
  - Combination of join methods for n-pass joins

- Costs of join evaluation depend on sequence of joins, but the final result does not

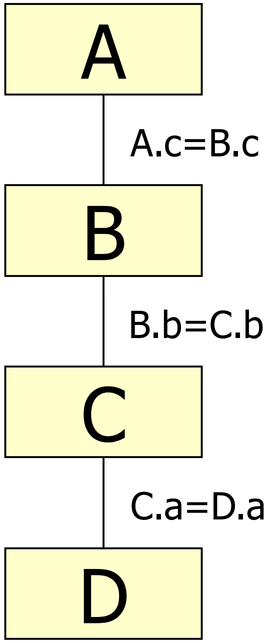
- Examination of all possible sequences of joins (exponential!) is avoided by a heuristic
  - Do not consider join sequences which involve Cartesian products
  - If Cartesian products are necessary, do them as late as possible

Example

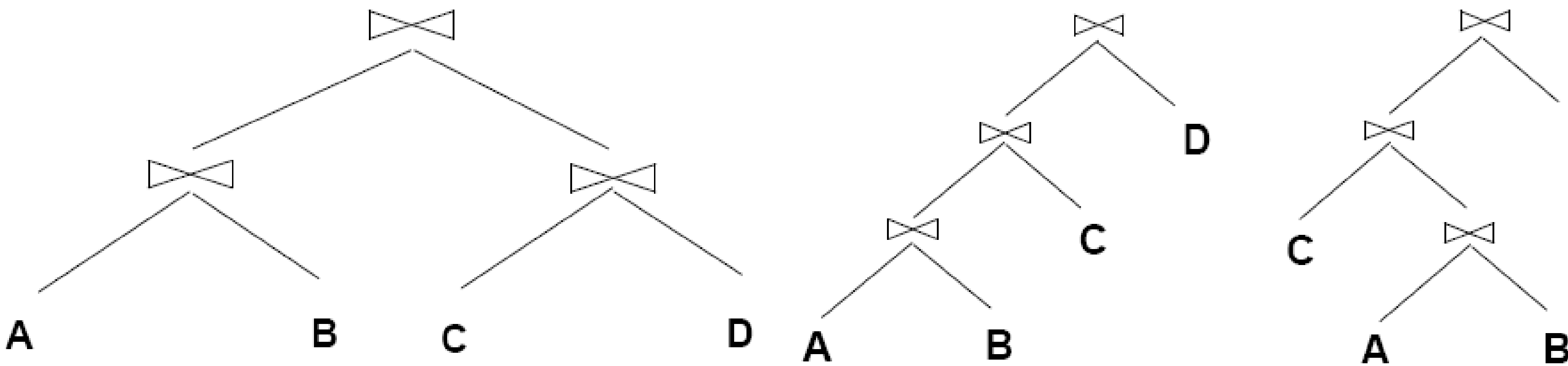
- How D is joined with $A \bowtie B \bowtie C$ is independent of the calculation of $A \bowtie B \bowtie C$ (unless result is sorted according join attribute C.a)

- If $A \bowtie B$ has been computed, it is not possible to join D in the next step ($A \bowtie B \bowtie D = A \bowtie B \times D$)

Quant Graph



Left-Deep Plans



Bushy

Left-Deep
(right is a base table)

Linear
(one child
is a base table)

In the following, we will just consider left-deep plans,
as most query optimizers in commercial DBMS do today.

Enumeration of Left-Deep Plans (1)

- **Left-deep plans differ only in:**
  - order of relations
  - access method for each relation
  - join method for each join

- **Enumerated using N passes (if N relations joined):**
  - **Pass 1:** Find best 1-relation plan for each relation.
  - **Pass 2:** Find best way to join result of each 1-relation plan (as outer) to another relation. (All 2-relation plans)
  - **Pass N:** Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. (All N-relation plans)

- **For each subset of relations, retain only:**
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

Enumeration of Left-Deep Plans (2)

- `ORDER BY, GROUP BY`, aggregates etc. handled as a final step, using either an *interestingly ordered* plan or an additional sorting operator.

- An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up
  – Avoid Cartesian products if possible

- This approach still requires exponential plan space in the number of tables
  – **Note:** This shows the need for semantic optimization. It is more efficient to avoid a join than to optimize its execution.

Example

Based on [Selinger et al., 1979]

```
SELECT e.name, t.tname, e.sal, d.dname
FROM EMP e, DEPT d, TASK t
WHERE t.tname='Design'
AND d.dname='SHIP'
AND e.dno=d.dno
AND e.eno=t.eno
```

DEPT

dno	dname	mgr
50	MFG	5
51	BILL	6
52	SHIP	7

EMPL

eno	name	dno	sal
3	Smith	50	85
4	Jones	50	150
5	Doe	51	95

TASK

eno	tname	pno
3	Design	34
4	Req.	33
5	Impl.	34
3	Design	46

1. Relevant single-relations-access plans:
   - each “interesting” order
   - Other orders if access is cheaper than cheapest “interesting” order

Example: Access Plans for Single Relations

- Scan of each relation
  - EMPL (sorted by eno)
  - DEPT (sorted by dno)
  - TASK (sorted by eno)
- Indexes:
  - EMPL.dno
  - EMPL.eno (clustered)
  - DEPT.dname
  - TASK.eno (clustered)
  - TASK.tname

- Required information
  - Costs (I/O + CPU, but we will just consider I/O costs in the following)
  - Size of result (after selection): #tuples and #pages
  - Order (if any)

Example: Costs for Access Plans

Access plan	Selection	Costs	#tuples	#pages	Order
Scan EMPL	-	B(EMPL)	T(EMPL)	B(EMPL)	eno
EMPL.dno	-	0.15*B(EMPL) +T(EMPL)	T(EMPL)	B(EMPL)	dno
EMPL.eno	-	(0.15+1.5)* B(EMPL)	T(EMPL)	B(EMPL)	eno
Scan DEPT	dname= 'SHIP'	B(DEPT)	F_D*T(DEPT)	F_D*B(DEPT)	dno
DEPT.dname	dname= 'SHIP'	0.15* B(DEPT)+ F_D*T(DEPT)	F_D*T(DEPT)	F_D* B(DEPT)	-
Scan TASK	tname= 'Design'	B(TASK)	F_T*T(TASK)	F_T*B(TASK)	eno
TASK.eno	tname= 'Design'	(0.15+1.5)* B(TASK)	F_T*T(TASK)	F_T*B(TASK)	eno
TASK.tname	tname= 'Design'	0.15*B(TASK) + F_T*T(TASK)	F_T*T(TASK)	F_T*B(TASK)	tname

F_T and F_D are the selectivity factors for the selection on TASK and DEPT

See also slide 31 in chapter 2

Example: Insert concrete values

B(EMPL)=500, T(EMPL)=40.000
B(TASK)=1000 T(TASK)=100.000
B(DEPT)=100 T(DEPT)=10.000
F_T=0.1; F_D =0.01

Worst case estimation, most likely cheaper in practice

Access plan	Selection	Costs	#tuples	#pages	Order
Scan EMPL	-	500	40.000	500	eno
EMPL.dno	-	40.075	40.000	500	dno
EMPL.eno	-	825	40.000	500	eno
Scan DEPT	dname='SHIP'	100	100	1	dno
DEPT.dname	dname='SHIP'	115	100	1	-
Scan TASK	tname='Design'	1.000	10.000	100	eno
TASK.eno	tname='Design'	1.650	10.000	100	eno
TASK.tname	tname='Design'	10.150	10.000	100	tname

Example: Remove non-optimal subplans

Access plan	Selection	Costs	#tuples	#pages	Order
Scan EMPL	-	500	40.000	500	eno
EMPL.dno	-	40.075	40.000	500	dno
~~EMPL.eno~~	~~-~~	~~825~~	~~40.000~~	~~500~~	~~eno~~
Scan DEPT	dname='SHIP'	100	100	1	dno
~~DEPT.dname~~	~~dname='SHIP'~~	~~115~~	~~100~~	~~1~~	-
Scan TASK	tname='Design'	1.000	10.000	100	eno
~~TASK.eno~~	~~tname='Design'~~	~~1.650~~	~~10.000~~	~~100~~	~~eno~~
~~TASK.tname~~	~~tname='Design'~~	~~10.150~~	~~10.000~~	~~100~~	~~tname~~

Example: Remaining plans

Relation	EMPL	EMPL	σ(DEPT)	σ(TASK)
Order	eno	dno	dno	eno
Pages	500	500	1	100
Tuples	40.000	40.000	100	10.000
Cost:	500	40.075	100	1000
	Scan EMPL	EMPL.dno	Scan DEPT	Scan TASK

Example: Two-Relation plans

2. Compute costs of all relevant join strategies for all remaining pairs of relations (avoiding cross products). Find the best results for
   - EMPL ⋈ TASK
   - EMPL ⋈ DEPT

$$X = \frac{\#tuples * tuple\ size}{page\ size}$$

Access plan	BNL	Merge	#tuples	#pages	Order
Scan EMPL⋈ Scan TASK	B(E)+B(E)* B(T')/(B-2)	B(E)+B(T')	F_{ET}*T(E)*T(T')	X	eno (Merge)
EMPL.dno⋈ Scan TASK	B(E)+B(E)* B(T')/(B-2)	B(E)+B(T')+ B(E)logB(E)	F_{ET}*T(E)*T(T')	X	eno (Merge)
Scan EMPL⋈ Scan DEPT	B(D')+B(D') *B(E)/(B-2)	B(E)+B(D')+ B(E)logB(E)	F_{ED}*T(E)*T(D')	X	dno (Merge)
EMPL.dno⋈ Scan DEPT	B(D')+B(D') *B(E)/(B-2)	B(E)+B(D')	F_{ED}*T(E)*T(D')	X	dno (Merge)

F_{ET} and F_{ED} are the selectivity factors for joins EMPL⋈TASK and EMPL⋈DEPT
T' and D' refer to the relations TASK and DEPT after selection

Example: Two-Relation plans with values

F_{ET}=1/40.000 F_{ED}=1/100 B=52

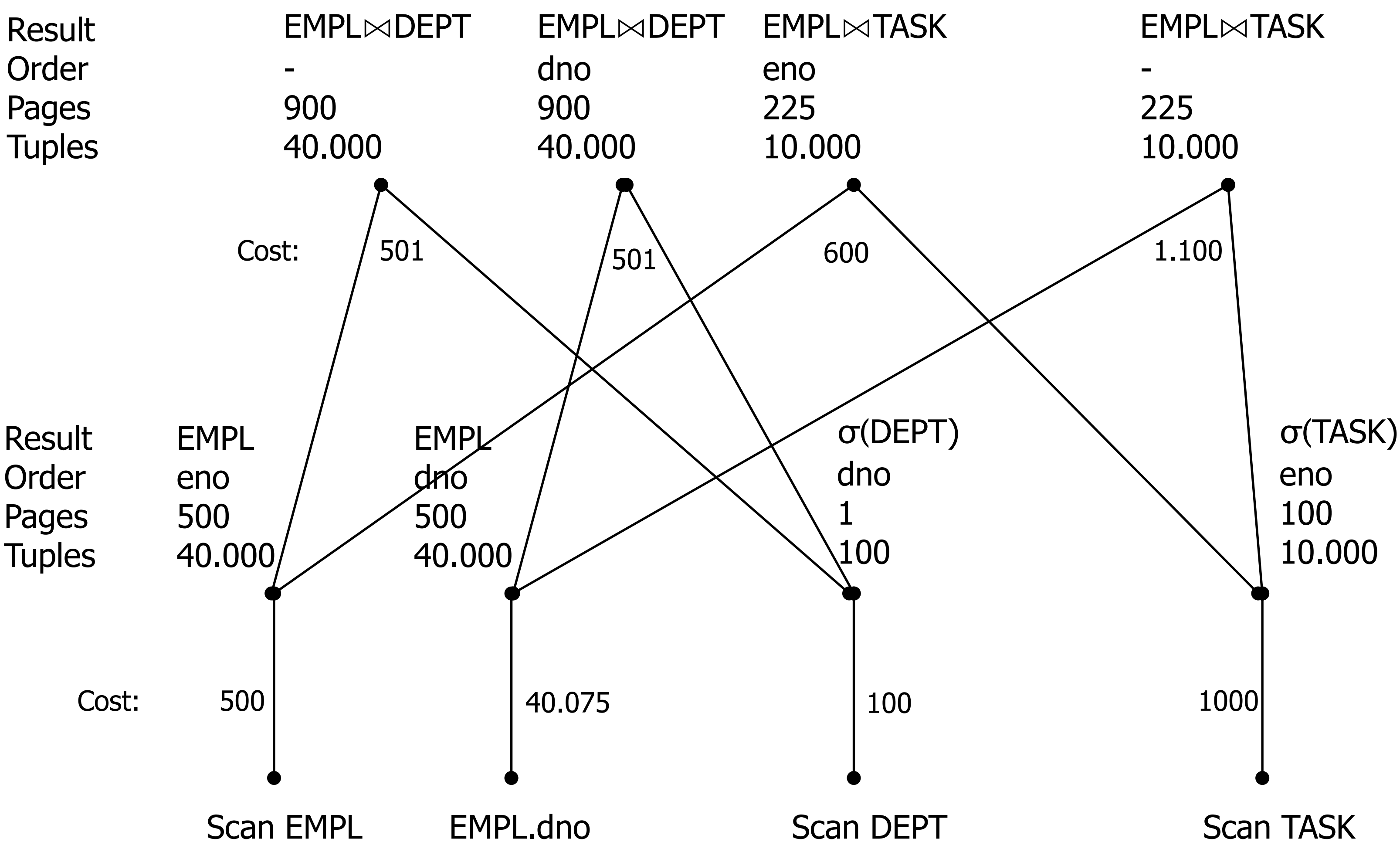
Assumption: Sorting for Merge join requires only one additional pass

Access plan	BNL	Merge	#tuples	#pages	Order of Best	Sub-plan	Best Total
Scan EMPL⋈ Scan TASK	1.500 (1)	600	10.000	225	eno	1.500	2.100
EMPL.dno⋈ Scan TASK	1.500 (1)	1.600	10.000	225	-	41.075	42.675
Scan EMPL⋈ Scan DEPT	501	1.501	40.000	900	-	600	1.101
EMPL.dno⋈ Scan DEPT	501	501	40.000	900	dno	40.175	40.676

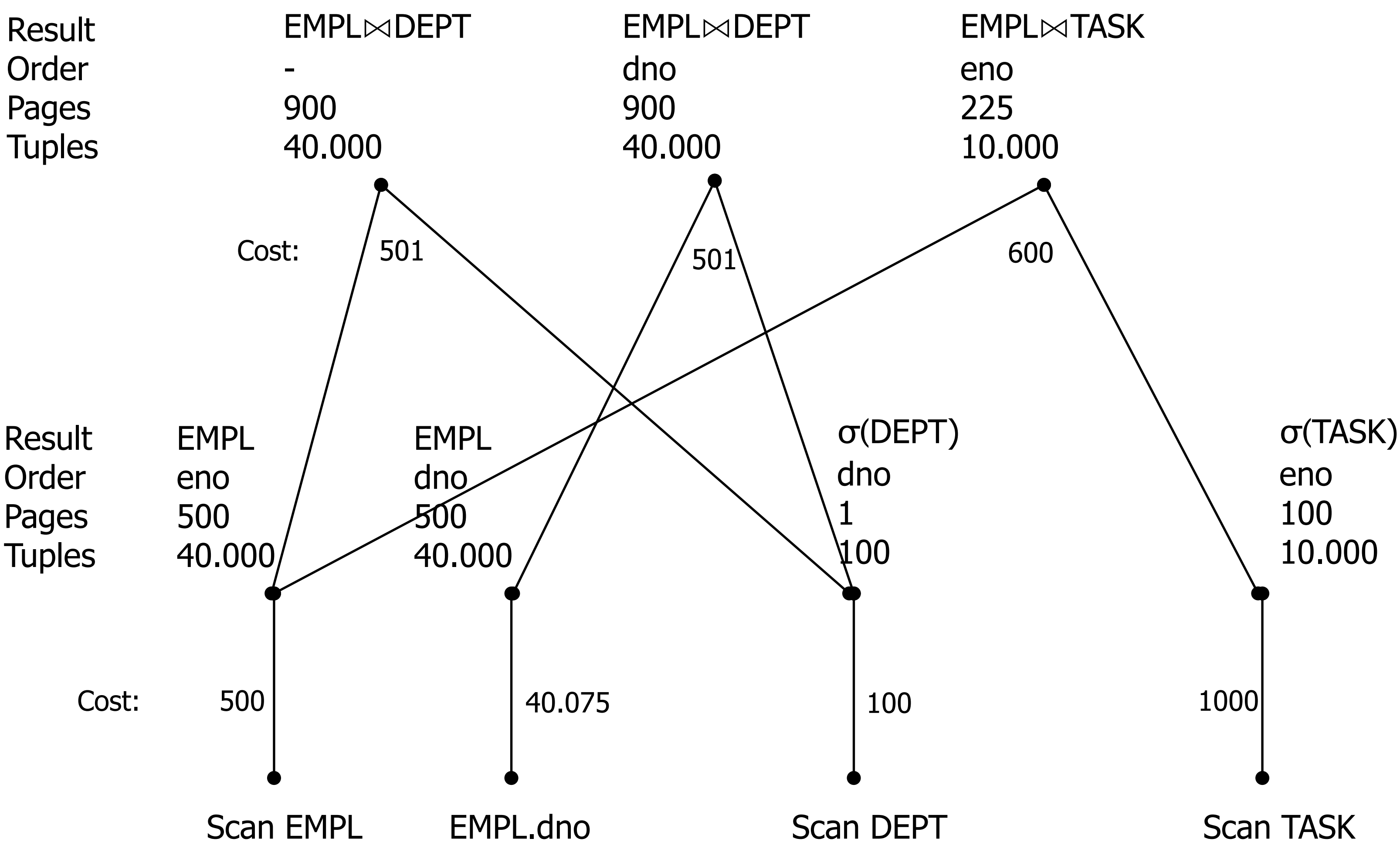
Plan with BNL join can be removed, because it does not preserve any order and is more expensive than the first plan. Plan with merge join preserves eno order as the first plan, but is also more expensive than the first. Therefore, it will be also removed.

(1) Is 1100 if TASK is the outer relation

Example: Graph for 2-relation plans



Example: Graph for remaining 2-relation plans



Example: 3-Relation plans

Access plan	BNL	Merge	Sub-plan	Best Total
(Scan EMPL $\bowtie_M$ Scan TASK) $\bowtie$ Scan DEPT	<u>226</u>	676	2.100	2.326
(Scan EMPL $\bowtie_B$ Scan DEPT) $\bowtie$ Scan TASK	<u>1.900</u>	2.800	1.101	3.001
(EMPL.dno $\bowtie_M$ Scan DEPT) $\bowtie$ Scan TASK	<u>1.900</u>	2.800	40.676	42.576

➔ **and the winner is …**: Merge join for joining EMPL and TASK (using relation scans), and then BNL join with DEPT

Summary: System R Approach

- Cost model based on
  - access methods
  - size and cardinality of relations
- Enumeration exploits
  - dynamic programming
  - one optimal plan for each equivalent expression (taking into account interesting orders)

Limitations of System R Approach

- **Cost Model**
  - one aggregate number for every column (inaccurate)
  - independence assumption
- **Transformation**
  - limited to join ordering
- **Enumeration**
  - limited to single block queries

3.4 Query Optimization Today

- More accurate selectivity estimation: Histograms
- Transformations
- QO in Today‘s Systems

More accurate selectivity estimation

- Histograms
  - Data structure to approximate data distribution
  - Range of values is divided into subranges (<u>buckets</u>)
  - In each bucket, number of tuples with a value in that range is counted
  - Equi-width histogram:
    - Subranges have equal sizes
  - Equi-depth histogram:
    - Number of tuples in each range is equal (or at least similar)
  - V-optimal histogram:
    - Minimize variance of frequency approximation
    - Buckets for “outliers”

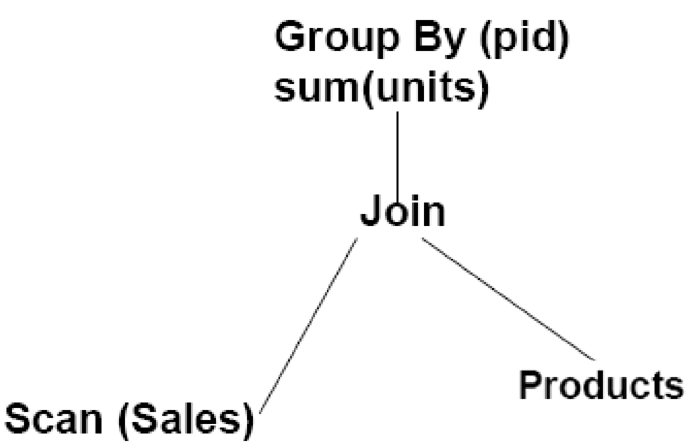
- **Advantage:** Optimization sensitive to available statistics

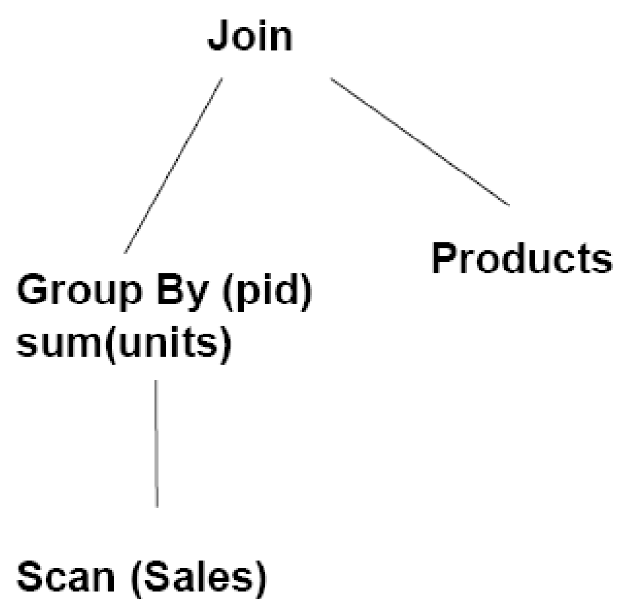
- **Disadvantage:** Expensive to collect and maintain

Transformations

- "Group by" usually executed at the end after all Join operations
- **Example:** Product(pid,price,…)
  Sales(tid,date, store, pid, units)
- **Query:**

```
SELECT pid, SUM(units)
FROM Product p, Sales s
WHERE p.pid = s.pid
GROUP BY p.pid
```





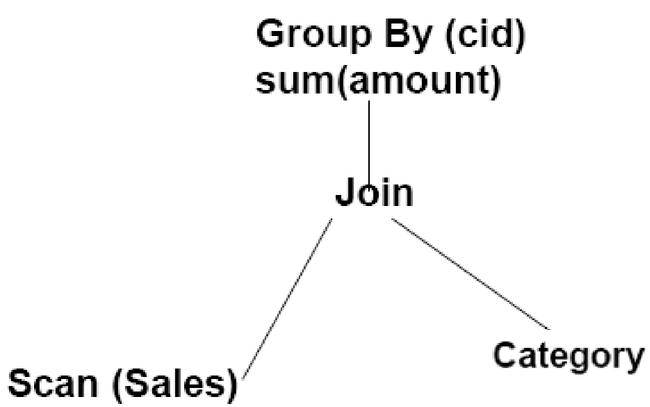
Introduction of Additional Operators

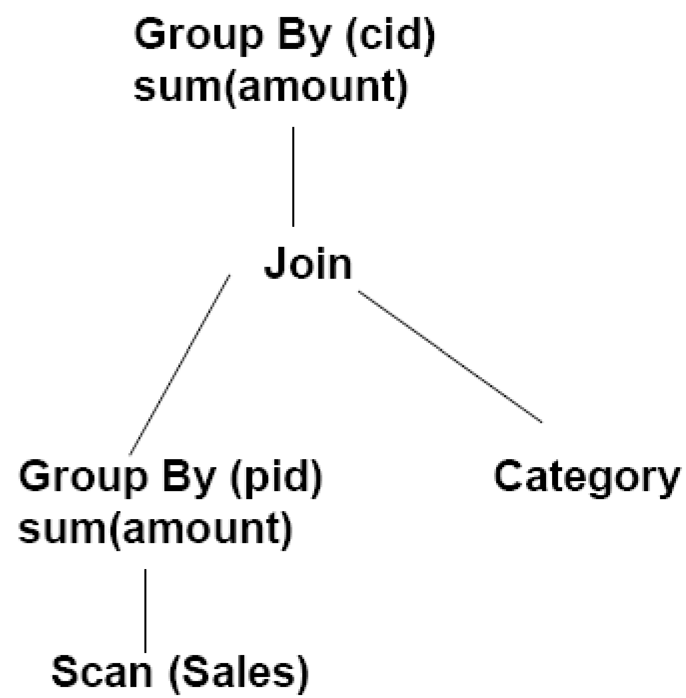
- Introducing Operators which reduce the size of intermediate result might reduce costs for subsequent join operations

- **Example:** Sales(tid,date,store,pid,units)
  Category(cid,pid)

- **Query:**

```
SELECT cid, SUM(units)
FROM Sales s, Category c
WHERE s.pid=c.pid
GROUP BY c.cid
```





QO in Today's Systems

- **Projection:**
  - Hash-based and Sort-Based implementations
- **Joins:**
  - PNL, INL, BNL, Hash, Sort-Merge
- **Estimation of Query Costs:**
  - Histograms (equi-depth with some variations)
  - Sampling for building histograms
- **Index-Only Queries:**
  - DBMS try to build index-only plans
  - Include columns: Columns to be included in index, but not part of key
- **Query Optimization:**
  - Dynamic Programming with left-deep plans with some variations
  - User might control optimizer, e.g. define join order or edit query execution plan

3.5 Views and Indexes

Long term “investment” in access paths

- Store partial results of queries as *index* or *materialized view* in the database
- **Advantage:** Queries can access pre-computed results
- **Disadvantages:**
  - Indexes & Views have to be maintained (higher update costs!)
  - Query processing needs to take into account views & indexes

Definition of such access paths is the task of physical DB design!

Creating Indexes & Views in SQL

- **Index on relations (primary/secondary):**
  - `CREATE INDEX IndexOnRel ON` *Rel* $(A_1,\dots,A_n)$
- **View index:**
  - `CREATE VIEW V1 AS SELECT ... FROM ... WHERE ...`
  - `CREATE INDEX IV1 ON V1 ...`

⇒ Result of V1 is stored in database *(**Materialization**!)*

⇒ Result of V1 might be used to answer queries

Example from MS SQL Server Doc

Create view:

```
CREATE   VIEW V1
WITH   SCHEMABINDING
AS
SELECT SUM(UnitPrice*Quantity*(1.00-Discount)) AS Revenue,
  OrderDate, ProductID, COUNT_BIG(*) AS COUNT
FROM   dbo.[Order Details] od, dbo.Orders o
WHERE   od.OrderID=o.OrderID
GROUP BY   OrderDate, ProductID
```

Create index on the view:

```
CREATE UNIQUE CLUSTERED INDEX IV1 ON V1 (OrderDate,
  ProductID)
```

Example from MS SQL Server Doc

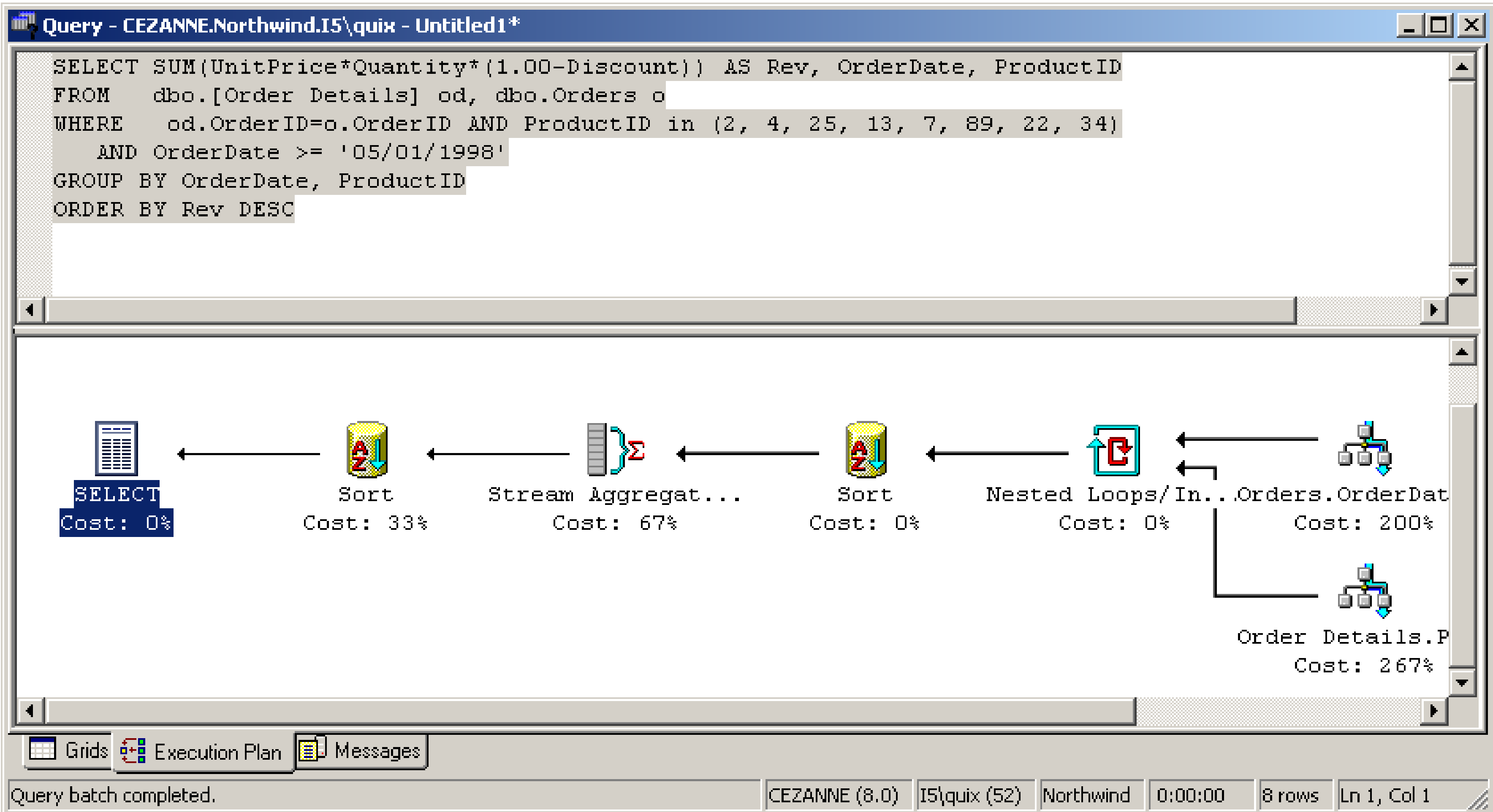
These queries will use the above indexed view:

```
SELECT SUM(UnitPrice*Quantity*(1.00-Discount)) AS Rev, OrderDate,
   ProductID
FROM   dbo.[Order Details] od, dbo.Orders o
WHERE   od.OrderID=o.OrderID AND ProductID in (2, 4, 25,...)
       AND OrderDate >= '05/01/1998'
GROUP BY OrderDate, ProductID
ORDER BY Rev DESC
```

```
SELECT  OrderDate, SUM(UnitPrice*Quantity*(1.00-Discount)) AS Rev
FROM   dbo.[Order Details] od, dbo.Orders o
WHERE   od.OrderID=o.OrderID AND DATEPART(mm,OrderDate)= 3
                               AND DATEPART(yy,OrderDate) = 1998
GROUP BY OrderDate
ORDER BY OrderDate ASC
```

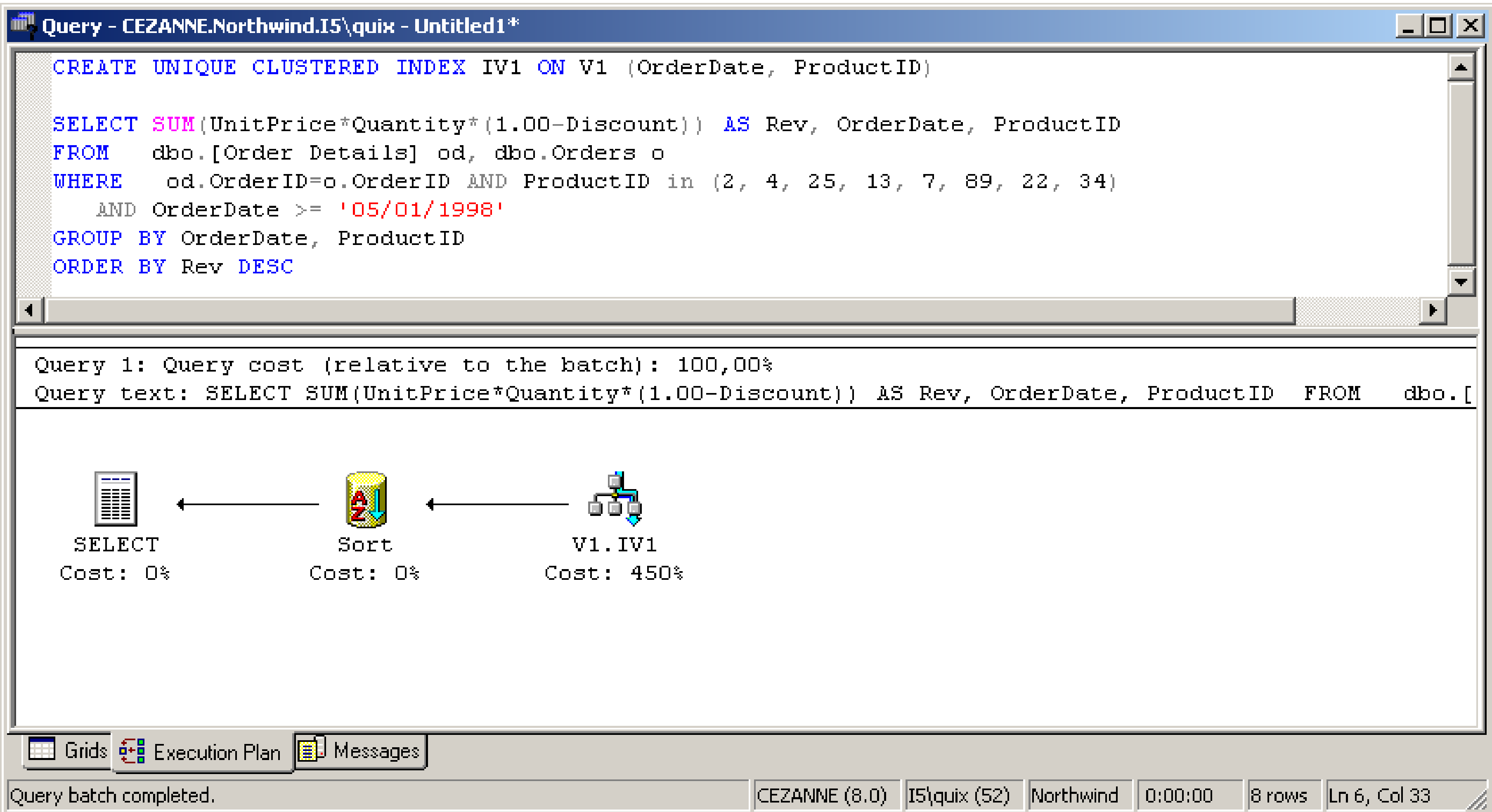
Query Execution Plan

Without indexed view!



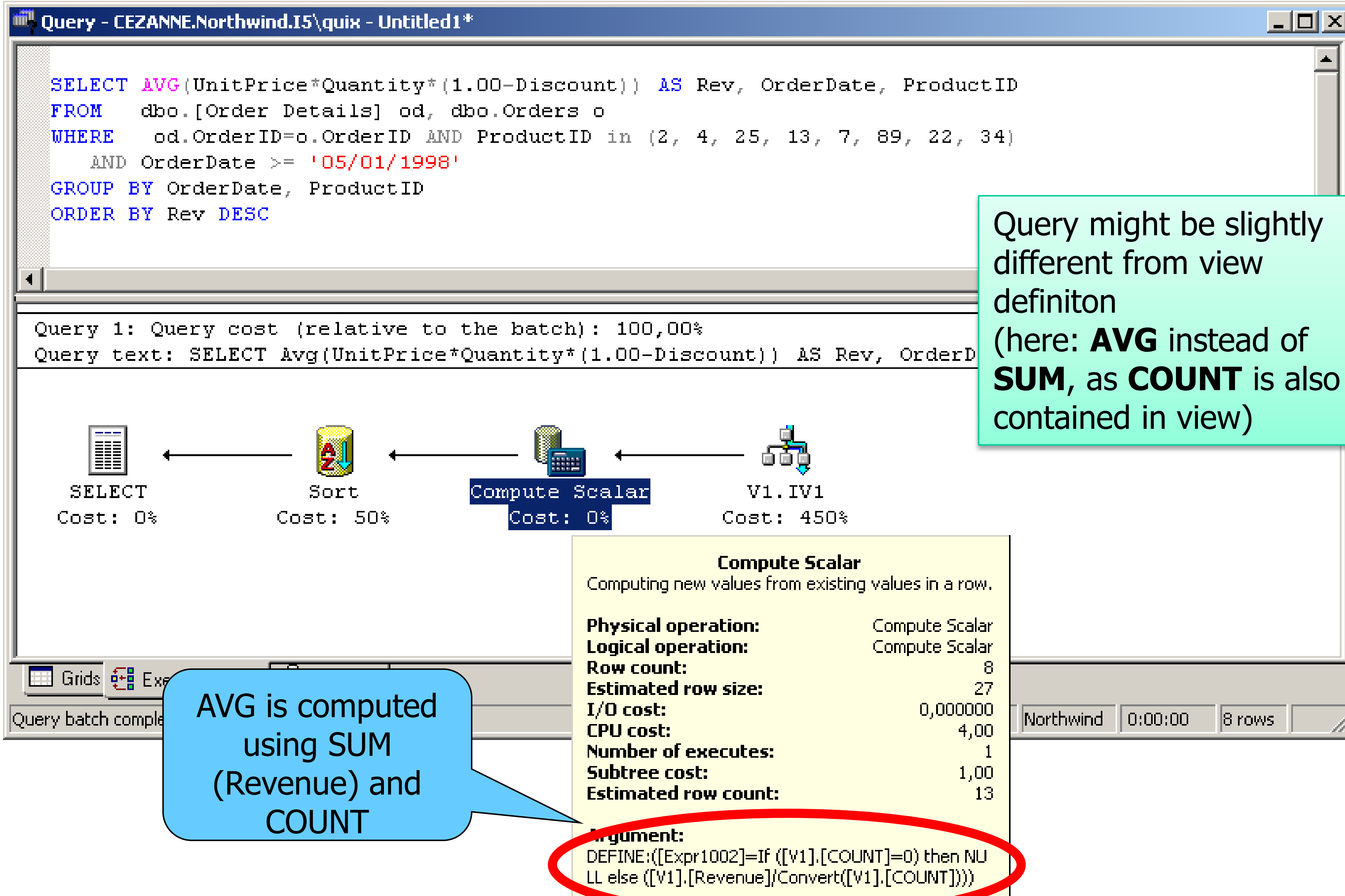
Screenshot of Query Analyzer of MS SQL Server 2000

With indexed view!



Screenshot of Query Analyzer of MS SQL Server 2000

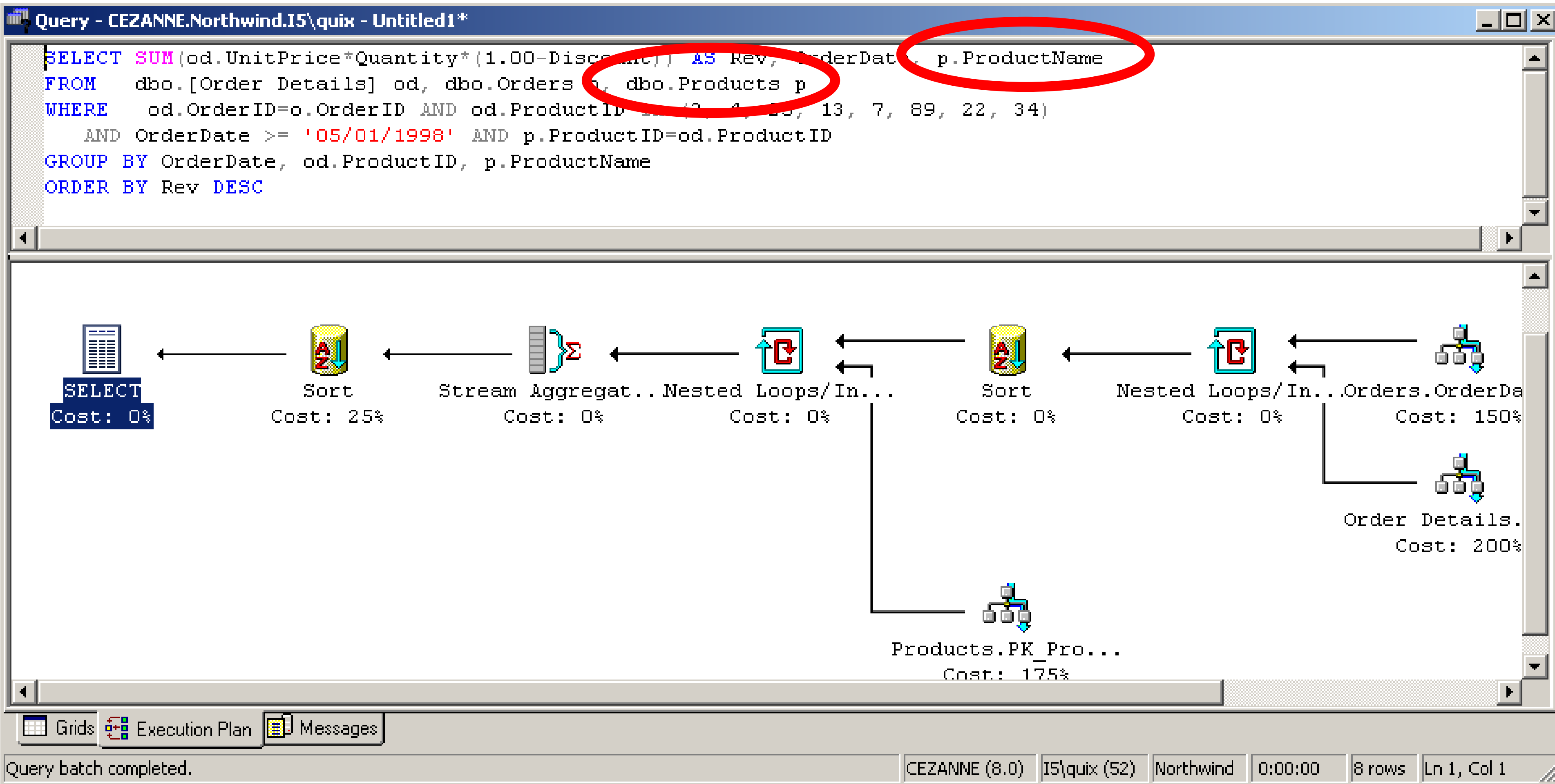
Query Execution Plan



Query might be slightly different from view definiton (here: **AVG** instead of **SUM**, as **COUNT** is also contained in view)

AVG is computed using SUM (Revenue) and COUNT

Back-Joins are not considered



Summary 3.4

- **Semantic and structure-based query optimization can reduce the complexity of query expressions**
  – Such techniques are important for system-generated queries (e.g., queries generated by a data access layer such as ADO.NET or Hibernate)

- **Exponential number of possible query plans**
  – Heuristic optimization (e.g., selection before join)
  – Dynamic programming

- **Accurate statistics important for cost estimation**
  ➔ Histograms

- **Create indexes and materialized views to speed up query execution**
  ➔ Index-only plans
  ➔ Database monitoring & tuning

Review Questions

- Which types of query optimization did we consider?
- What is the difference between semantic and structural query optimization?
- What is the goal of the tableau method?
- What is the principle of optimality in dynamic programming? How is it applied in query optimization?
- Which strategies are applied to reduce the number of query plans to be considered in cost-based query opt.?
- Which sub-plans will be maintained for the next stage in cost-based query optimization?
- Why do we need selectivity esitmation? What is a histogram?
- How to improve the query performance in a RDBMS?

References

[Abiteboul et al., 1995] Abiteboul, S.; Hull, R.B.; Vianu, V.: Foundations of Databases. Addison-Wesley, 1995. http://webdam.inria.fr/Alice/

[Aho et al., 1979] Aho, A. V.; Sagiv, Y. & Ullman, J. D.: Efficient optimization of a class of relational expressions. ACM Transactions on Database Systems (TODS), ACM, 1979, 4, 435-454.

[Jarke & Koch, 1983] Jarke, M. & Koch, J.: Range nesting: A fast method to evaluate quantified queries. ACM SIGMOD Record, 1983, 13, 196-206.

[Jarke & Koch, 1984] Jarke, M. & Koch, J.: Query optimization in database systems. ACM Computing surveys (CsUR), ACM, 1984, 16, 111-152.

[Selinger et al., 1979] Selinger, P. G.; Astrahan, M. M.; Chamberlin, D. D.; Lorie, R. A. & Price, T. G.: Access path selection in a relational database management system. Proceedings of the 1979 ACM SIGMOD international conference on Management of data, 1979, 23-34.