DATABASES

Query Processing

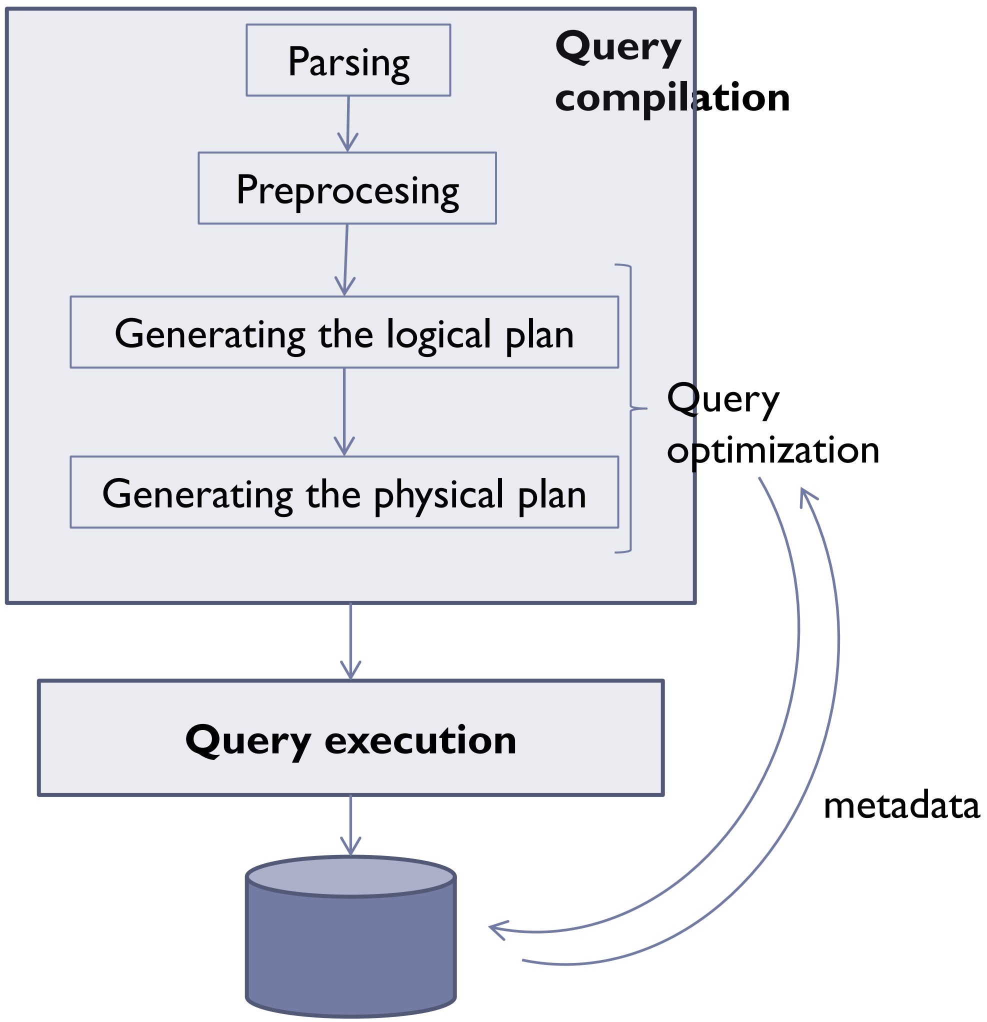
Mihaela Elena Breabăn

Outline

- Main phases of Query Processing
- Expressions in relational algebra
  - Operators (revisited)
  - Expressions
  - Equivalence of expressions
- Estimating the cost of a query
- Algorithms for processing the relational operators
- Oracle DBMS: execution plans, statistics, query hints

Steps in Query Processing

- Compiling the query
  - Syntactic analysis
    - Parsing
      - Parsing tree
  - Semantic analysis
    - Preprocessing and rewriting in RA
    - Selection of the relational algebraic representation
      - Logical plan
    - Selection of the algorithms
      - Physical plan
- Executing the physical plan



I. Syntactic analysis

- Context-free grammar

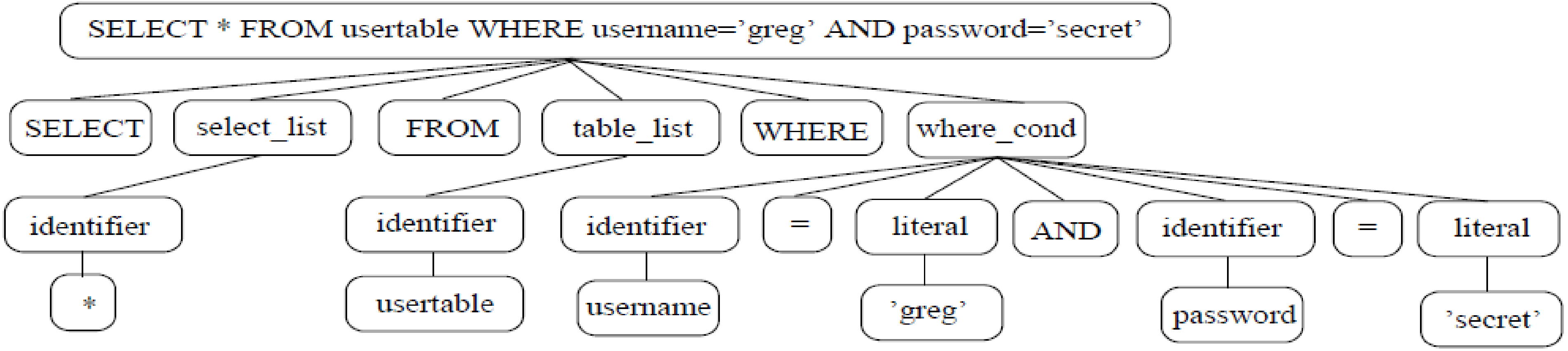
  <query> ::= <SFW> | (<query>)

  <SFW> ::= SELECT <select_list> FROM <table_list> WHERE <where_cond>

  <select_list> ::= <identifier>, <select_list> | <identifier>

  <table_list> ::= <identifier>, <table_list> | <identifier >

  …
- Parsing result: parsing tree



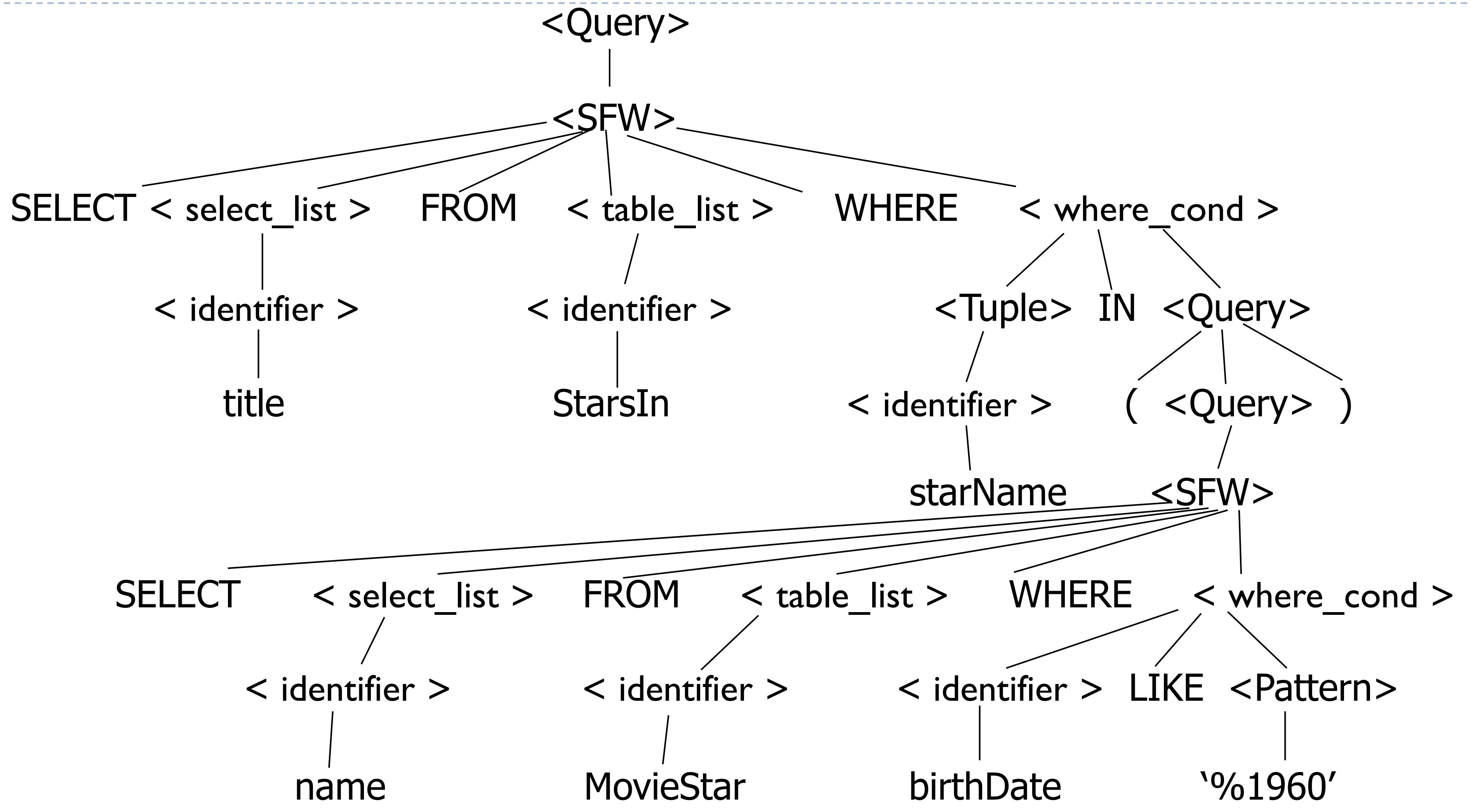
- Example of SQL grammar in BNF: http://savage.net.au/SQL/index.html

II. Semantic analysis
a. Preprocessing

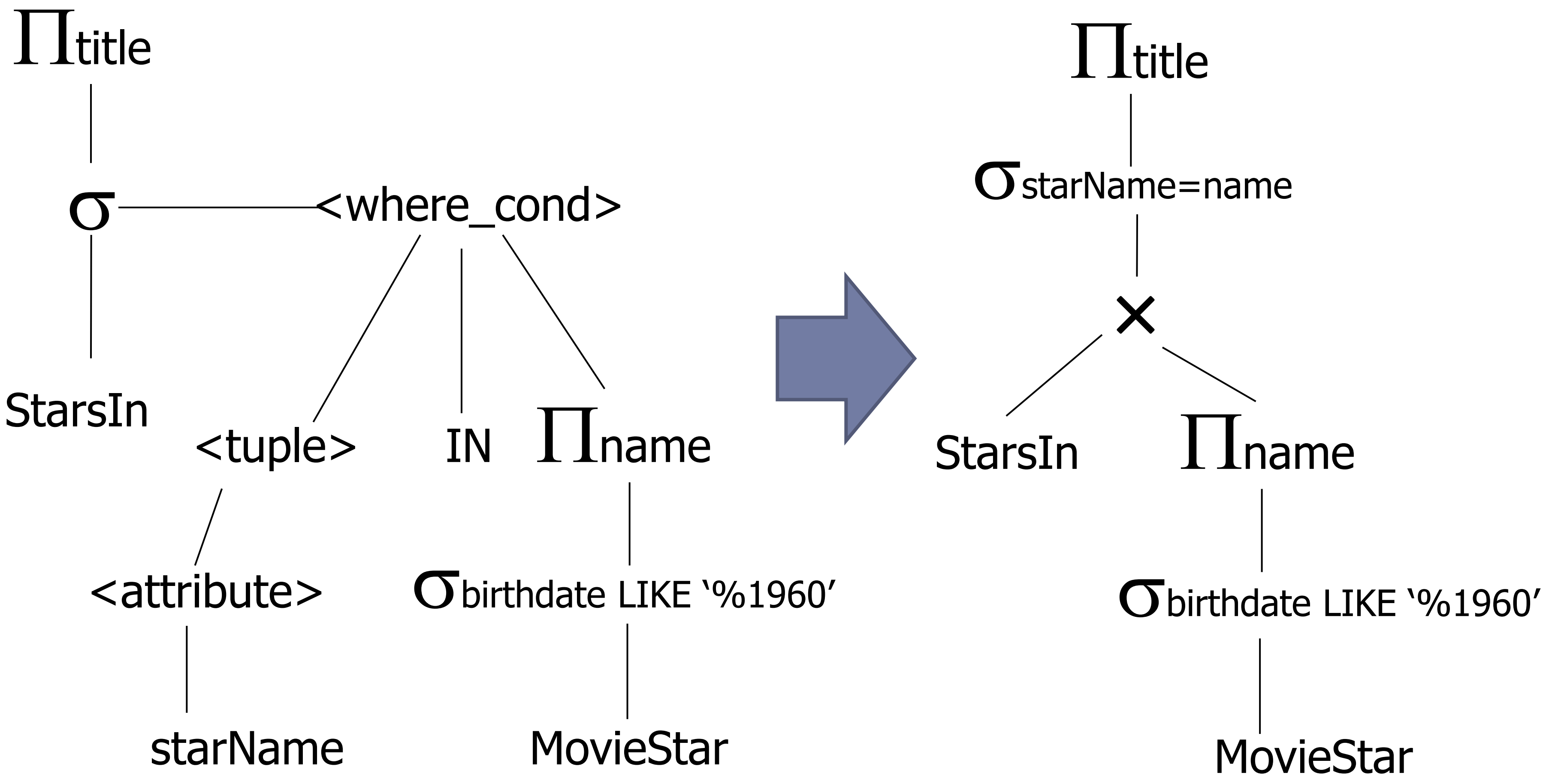
- Rewrite calls to views
- Verify existence of relations
- Verify existence of attributes and ambiguity
- Verify data types

If the parsing tree is valid, it is transformed into an expression in Relational Algebra (RA)

II. Semantic analysis
b. Rewriting in RA

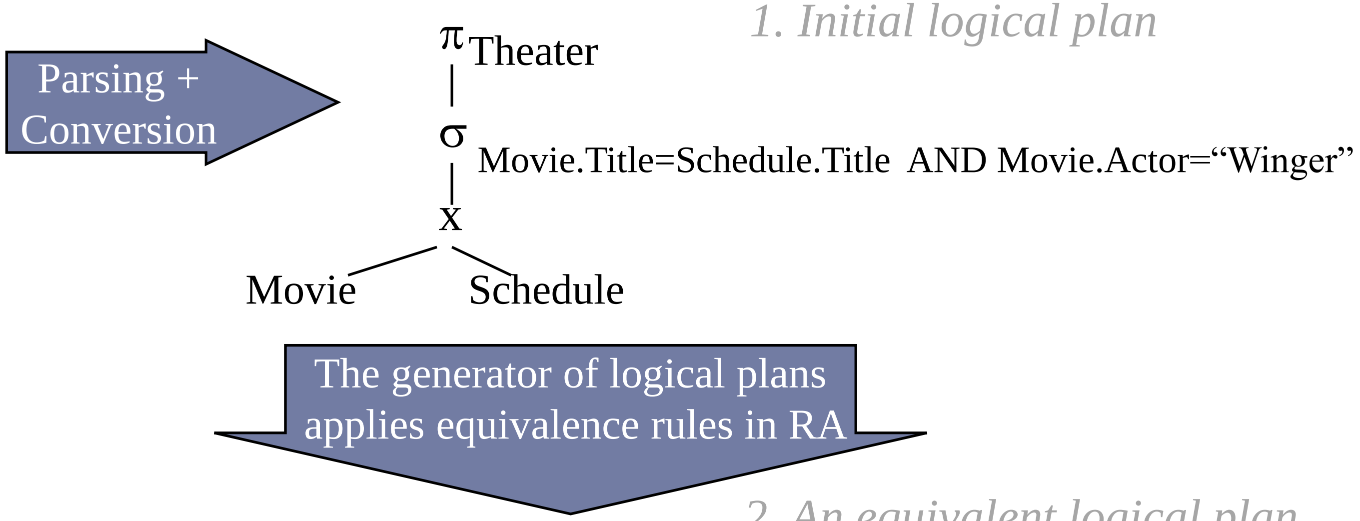


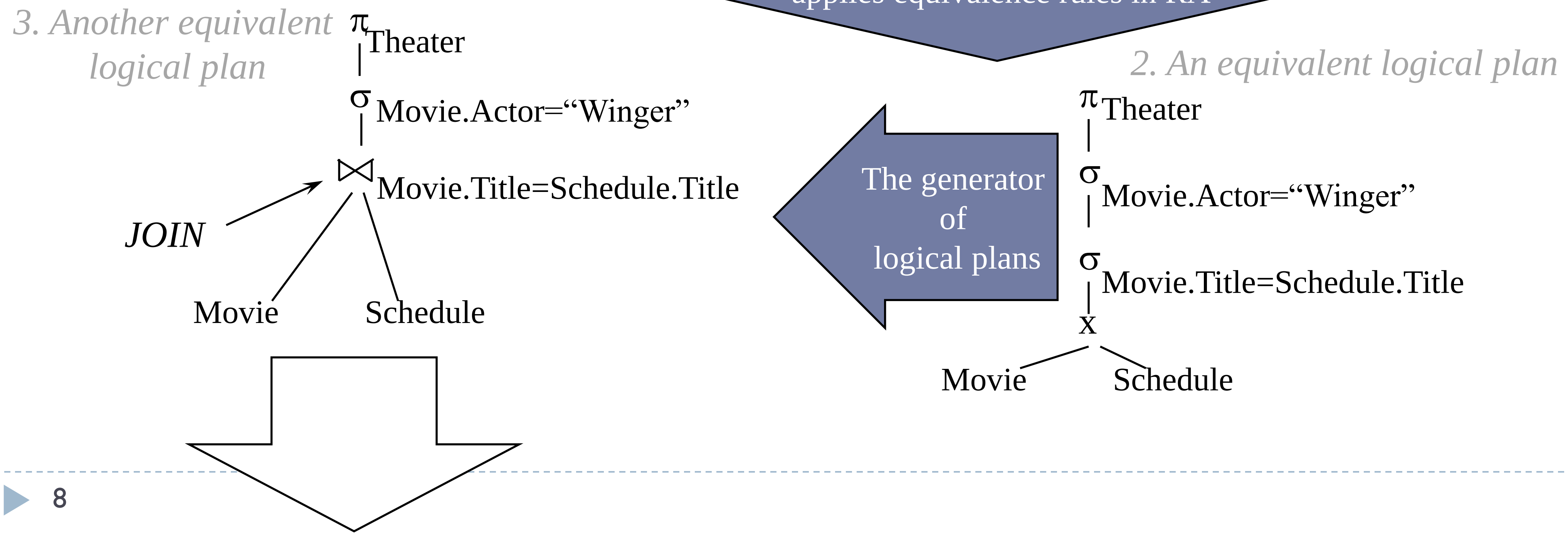
II. Semantic analysis
b. Rewriting in RA (continued)



II. Semantic analysis
c. Logical plan - optimization

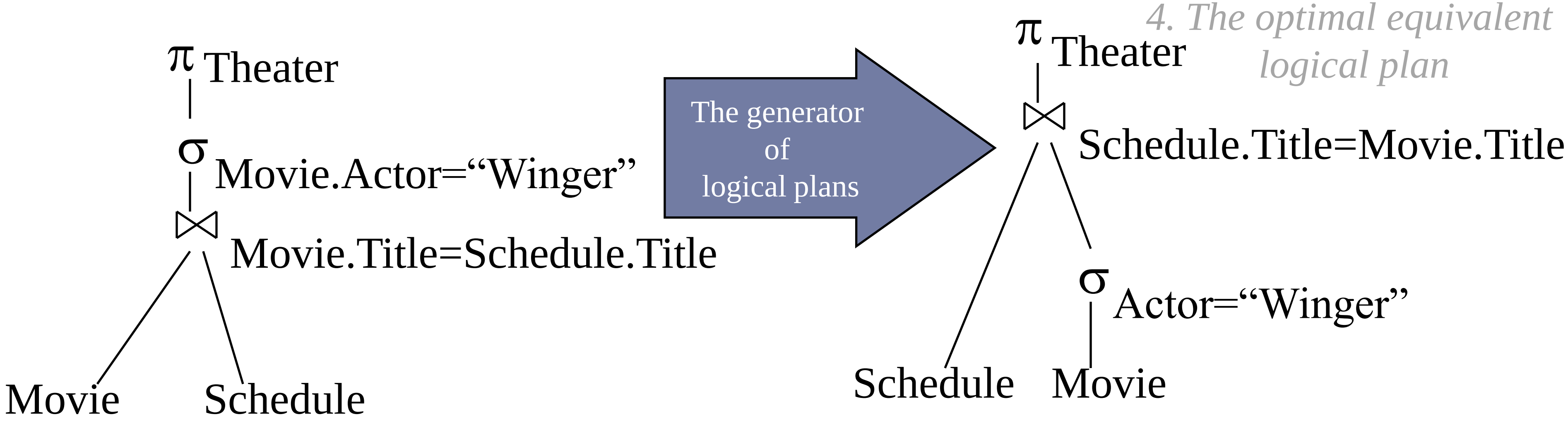
SELECT Theater
FROM Movie, Schedule
WHERE
Movie.Title = Schedule.Title
AND Movie.Actor="Winger"



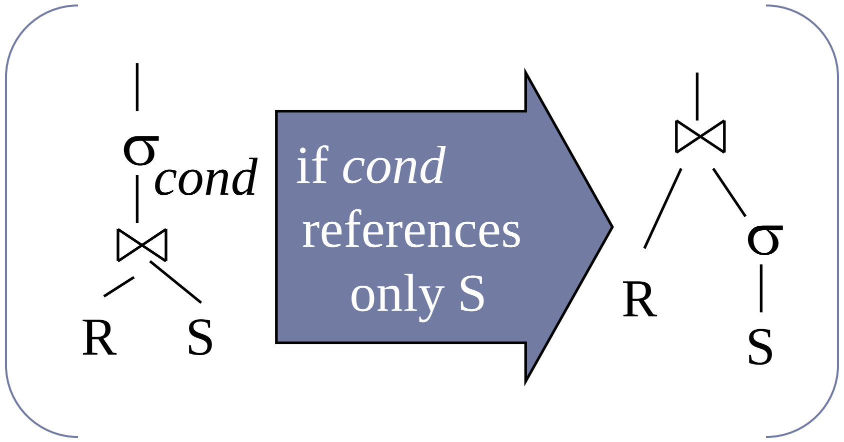


II. Semantic analysis
c. Logical plan – optimization (continued)

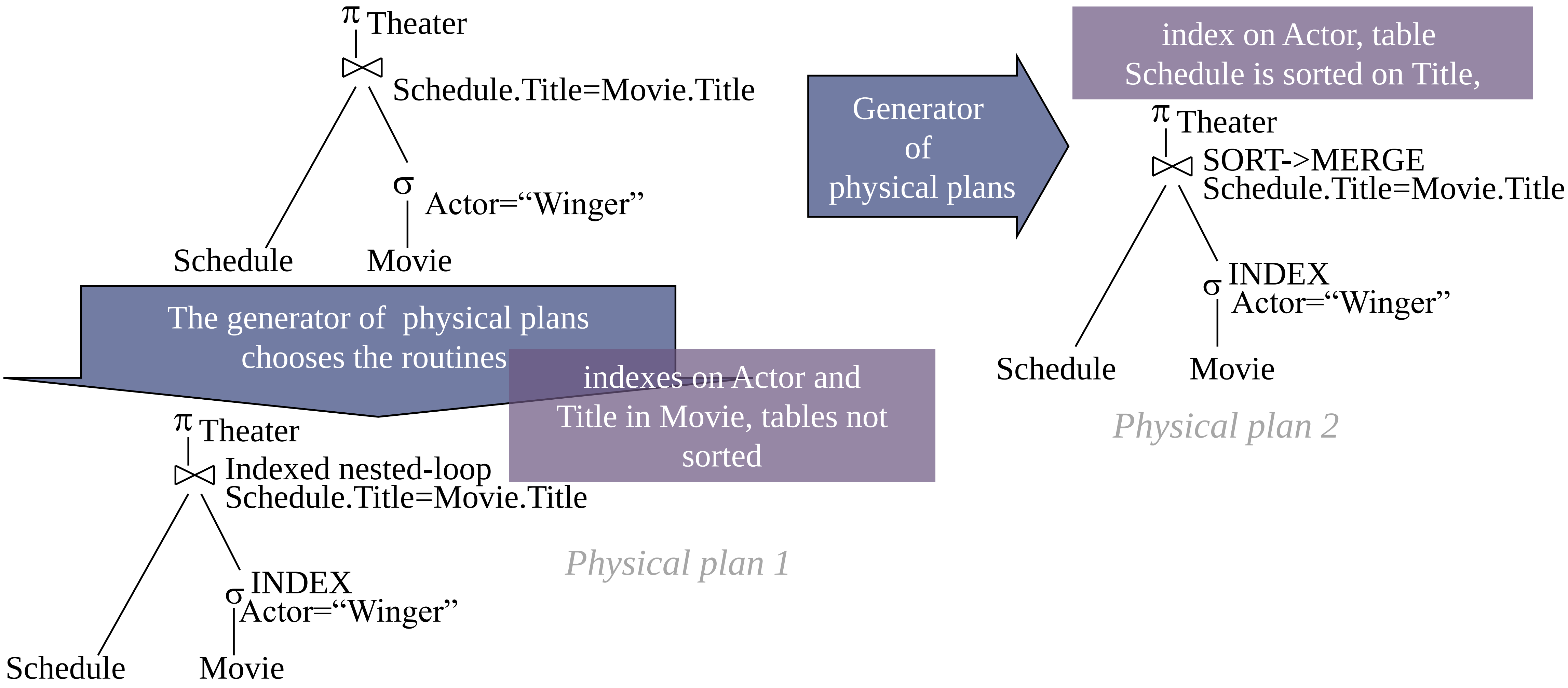
4. The optimal equivalent logical plan



Equivalence rule applied:



II. Semantic analysis
d. Physical plan - optimization



Operators in relational algebra (revisited)

- Six basic operators:
  - Selection: σ
  - Projection: $\prod$
  - Union: $\cup$
  - Set difference: $-$
  - Cartesian product: x
  - Renaming: ρ

- The operators act on one or two relations and generate one new relation

Selection

- r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}\,(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Projection

- r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\prod_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

A	C
α	1
β	1
β	2

Union

- r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r $\cup$ s:

A	B
α	1
α	2
β	1
β	3

Set difference

- r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r-s

A	B
α	1
β	1

Cartesian product

- r, s

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r x s*

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Renaming

- $\rho_X(E)$ - returns the result of expression E named as X

- If the result of expression E has n attributes than

$$\rho_{x(A_1,A_2,\ldots,A_n)}(E)$$

returns the result of E named as X with attributes renamed as $A_1, A_2, \ldots, A_n$.

Operators composition

- $\sigma_{A=C}(r \times s)$

1. $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

2. $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

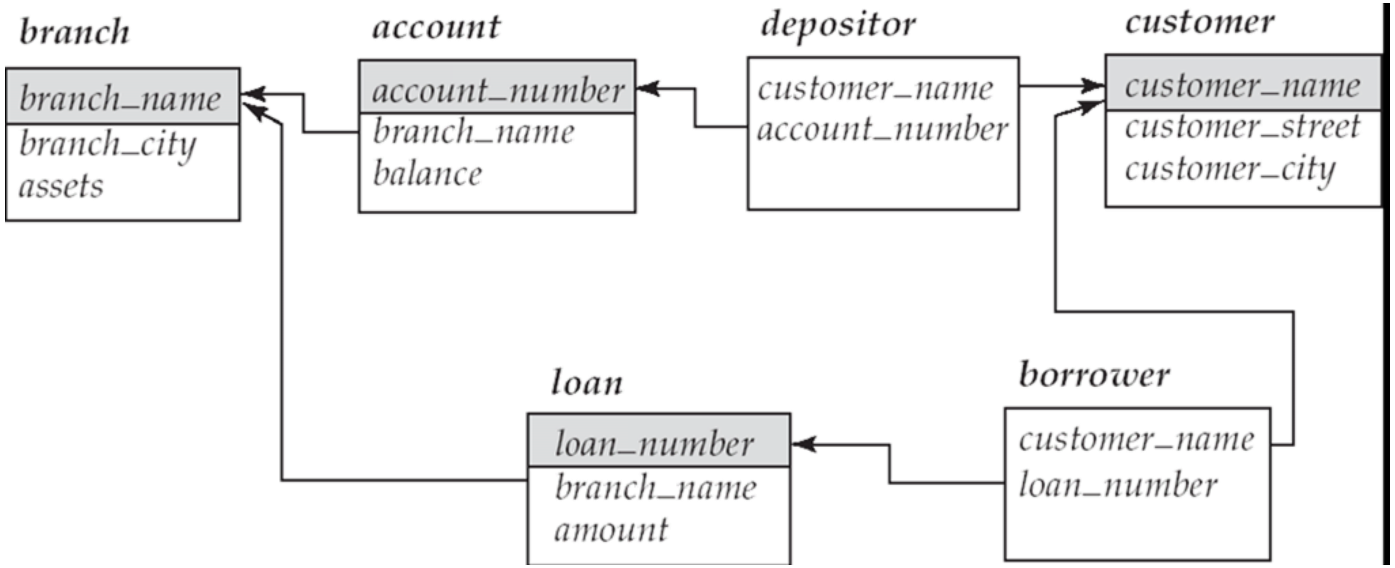
Expressions in relational algebra -a recursive definition

- The simplest expression is a relation *r*
- Let E_1 and E_2 be expressions in RA;

then, the following are also expressions in RA:

  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, *P* is a predicate over attributes in E_1
  - $\prod_s(E_1)$, *S* is a list of attributes in E_1
  - $\rho_x (E_1)$, x is a new name for E_1

Expressing queries in RA



- Loans greater than1200

$$\sigma_{amount > 1200}\ (loan)$$

- Loan number for loans greater than 1200

$$\Pi_{loan\_number}\ (\sigma_{amount > 1200}\ (loan))$$

- Name of the clients with a loan, a deposit or both

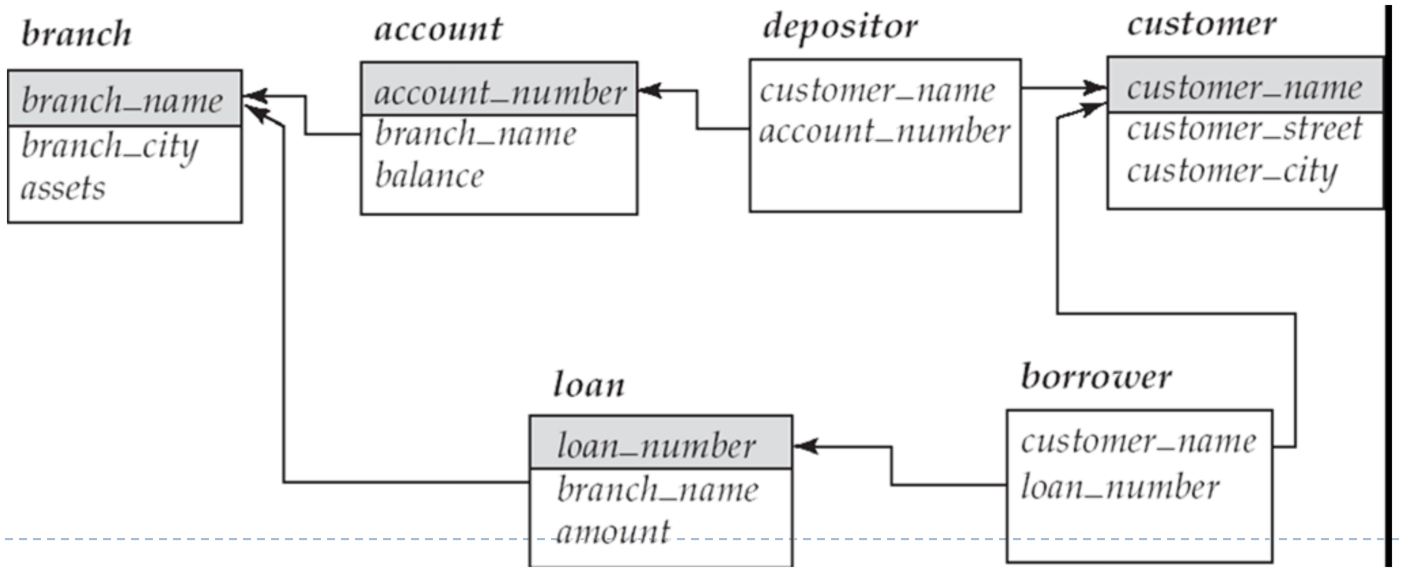
$$\Pi_{customer\_name}\ (borrower) \cup \Pi_{customer\_name}\ (depositor)$$

Expressing queries in RA (ctd.)

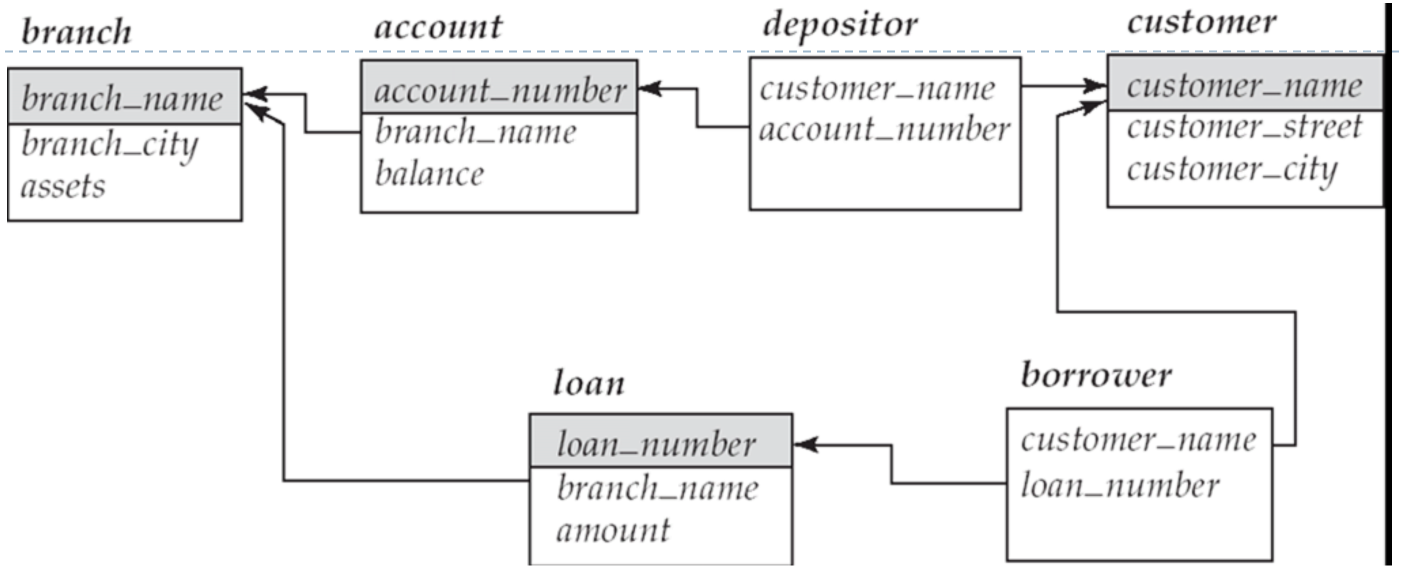
- Name for the clients having loans at the Perryridge branch

  - $\Pi_{customer\_name}\,(\sigma_{branch\_name = \text{"Perryridge"}}\,(\sigma_{borrower.loan\_number\, =\, loan.loan\_number}\,(borrower \times loan)))$

  - $\Pi_{customer\_name}(\sigma_{loan.loan\_number\, =\, borrower.loan\_number}\,((\sigma_{branch\_name = \text{"Perryridge"}}(loan)) \times borrower))$



Expressing queries in RA (ctd.)



- Name for the clients having loans at the Perryridge branch but having no deposits

$$\Pi_{customer\_name}\,(\sigma_{branch\_name\,=\,\text{“Perryridge”}}\,(\sigma_{borrower.loan\_number\,=\,loan.loan\_number}(\text{borrower x loan})))$$
$$-\quad \Pi_{customer\_name}(\text{depositor})$$

Additional relational operators

- Set intersection
- Natural join
- Aggregation
- External join
- Theta-join

- All of them, excepting aggregation, can be expressed using basic operators

Set intersection

- r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Natural join

- r, s

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

- $r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

- $\Pi_{r.A,\, r.B,\, r.C,\, r.D,\, s.E}\ (\sigma_{r.B\, =\, s.B\ \wedge\ r.D\, =\, s.D}\ (r\ \times\ s))$

Aggregation

- Functions:
  - **avg**
  - **min**
  - **max**
  - **sum**
  - **count**
  - **var**

- Syntax:

$$_{G_1, G_2, \ldots, G_n}\vartheta_{F_1(A_1), F_2(A_2), \ldots, F_n(A_n)}(E)$$

  - E – expresion in RA
  - $G_1, G_2 \ldots, G_n$ a list of grouping attributes (may be empty)
  - Every F_i is an aggregation function
  - Every A_i is an attribute

Aggregation Example

- r

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

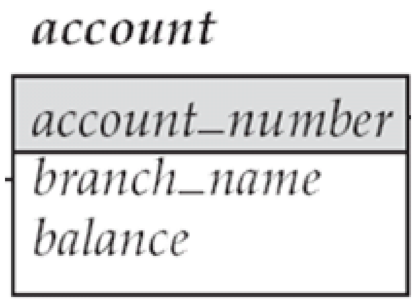
- $\mathcal{G}_{\textbf{sum(c)}}(\mathrm{r})$

sum(c)
27

- Which aggregation functions may be expressed based on basic relational operators?

Aggregation
Example using basic operators

- The largest balance in the account table



$$\Pi_{balance}(account) - \Pi_{account.balance}\left(\sigma_{account.balance < d.balance}\left(account \times \rho_{d}(account)\right)\right)$$

External join

loan

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

- *loan* $\bowtie$ *borrower (natural join)*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- *loan* $⟕$ *borrower (left external join)*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

External join

- right external join

loan ⟖ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- full external join

loan ⟗ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Expressing queries in RA more examples

- Name for the clients having both a loan and a deposit

$$\Pi_{customer\_name}\ (borrower) \cap \Pi_{customer\_name}\ (depositor)$$

- Name for the clients having a loan and the amount

$$\Pi_{customer\_name,\ lmount}\ (borrower \bowtie loan)$$

- Clients having deposits at at least the two branches named Downtown and Uptown

$$\Pi_{customer\_name}\ (\sigma_{branch\_name\ =\ \text{"Downtown"}}\ (depositor \bowtie\ account\ )) \cap$$
$$\Pi_{customer\_name}\ (\sigma_{branch\_name\ =\ \text{"Uptown"}}\ (depositor \bowtie account))$$

Equivalence of expressions
Definition

- Two expresions in RA are *equivalent* if they generate the same set of tuples on any instance of the database
  - Remember: the order of tuples is not relevant

- Obs: SQL works with multisets

Equivalence Rules

1. selection based on conjunctions is equivalent with a sequence of selections

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. selections are comutative

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. in a sequence of projections only the last one is necessary

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L_1}(E)$$

4. selections may be combined with the cartesian product

   a. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$
   b. $\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$

Equivalence Rules

5. theta-join and natural join are commutative

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

6. natural joins are associative

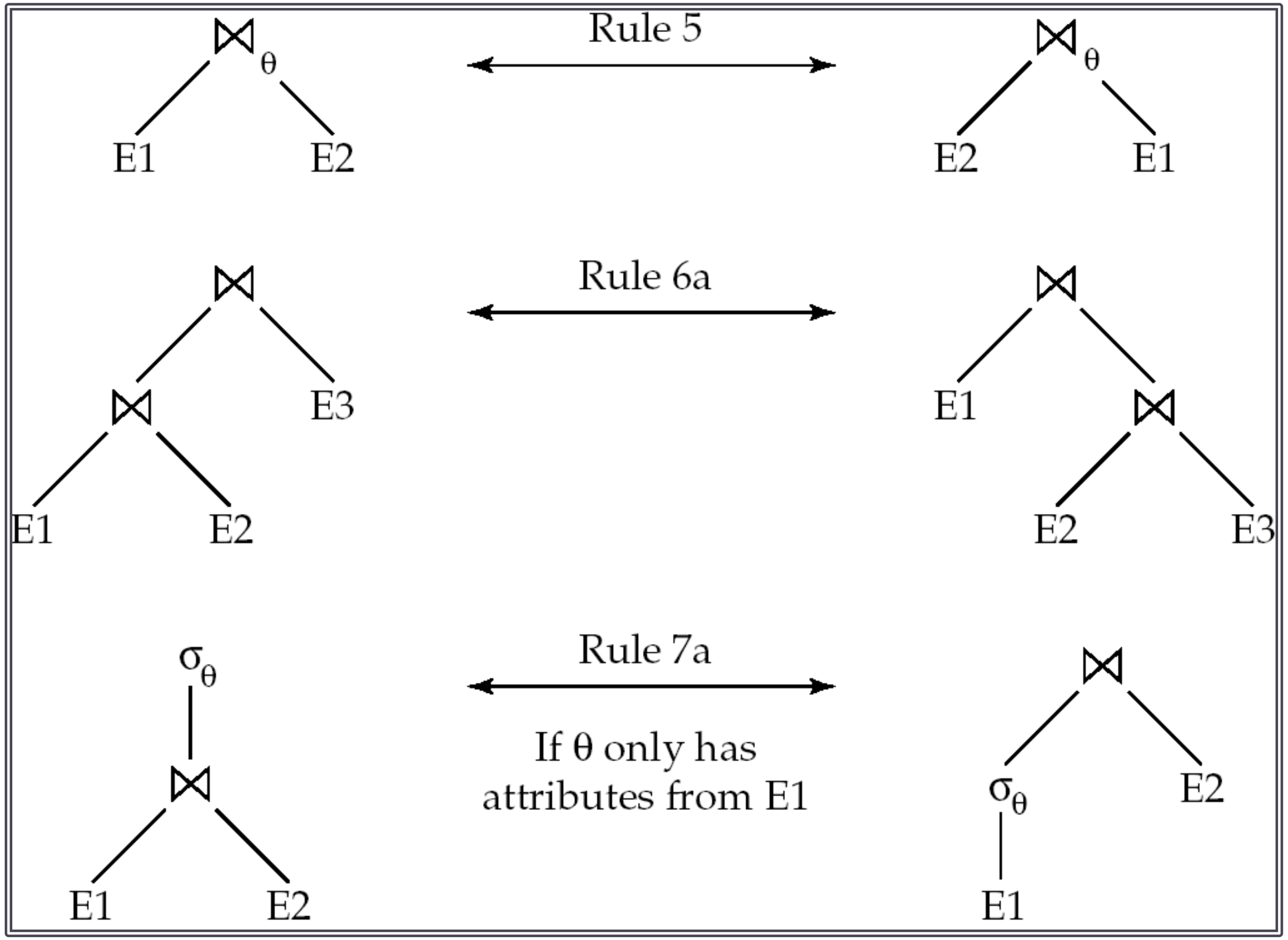
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

b) theta-joins are associative with some restrictions

$$(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$$

where θ_2 involves only attributes in E_2 and E_3

Equivalence Rules
- visualization



Equivalence Rules

7. selection may be distributed over theta-join
   a) when θ_0 involves only attributes in (E_1) :

$$\sigma_{\theta 0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 0}(E_1)) \bowtie_{\theta} E_2$$

   b) When θ involves only attributes in E_1 and θ_2 involves only attributes in E_2:

$$\sigma_{\theta 1 \wedge \theta 2} (E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 1}(E_1)) \bowtie_{\theta} (\sigma_{\theta 2} (E_2))$$

Equivalence Rules

8. **projection may be distributed over theta-join**
   a) If θ involves only attributes in $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

   b) Consider the join $E_1 \bowtie_\theta E_2$

Let L_1 and L_2 be sets of attributes in E_1 and E_2, respectively

Let $L3$ contain attributes in $E1$ involved in θ, but not in $L_1 \cup L_2$,

Let L_4 contain attributes in E_2 involved in θ, but not in $L_1 \cup L_2$

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

Equivalence Rules

9. set union and intersection are commutative

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

10. set union and intersection are associative

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. selection may be distributed over $\cup$, $\cap$ and $-$.

$$\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta(E_2)$$

similar for $\cup$ and $\cap$ instead of $-$

$$\sigma_\theta (E_1 - E_2) = \sigma_\theta(E_1) - E_2$$

similar for $\cap$ instead of $-$, but not for $\cup$

12. projection may be distributed over union

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

Logical plan optimization

Optimization
Pushing selection

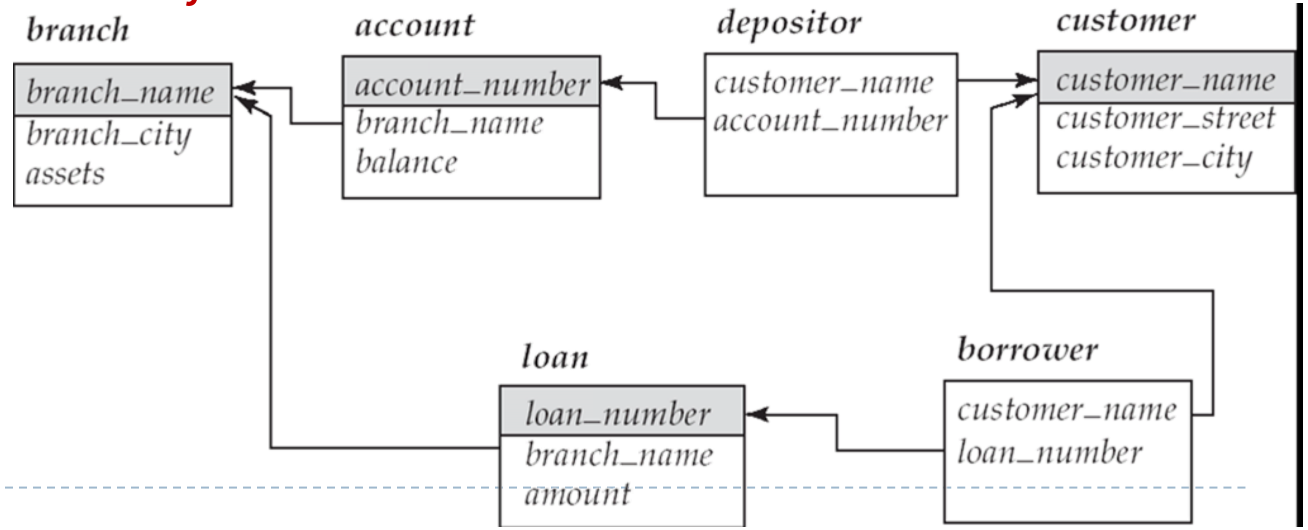
- Example 1:
  - Name of the clients having an account at the branches located in Brooklyn

$$\Pi_{customer\_name}(\sigma_{branch\_city\ =\ \text{“Brooklyn”}}(branch \bowtie (account \bowtie depositor)))$$

  - Based on rule 7a obtain:

$$\Pi_{customer\_name}((\sigma_{branch\_city\ =\text{“Brooklyn”}}(branch)) \bowtie (account \bowtie depositor))$$

- *By performing selection earlier, the size of the relations at join becomes smaller*



Optimization
Pushing selection

- Example 2:
  - Name of the clients having an account at the branches located in Brooklyn having the balance greater than1000

$$\Pi_{customer\_name}(\sigma_{branch\_city = \text{“Brooklyn”} \wedge balance > 1000}(branch \bowtie (account \bowtie depositor)))$$

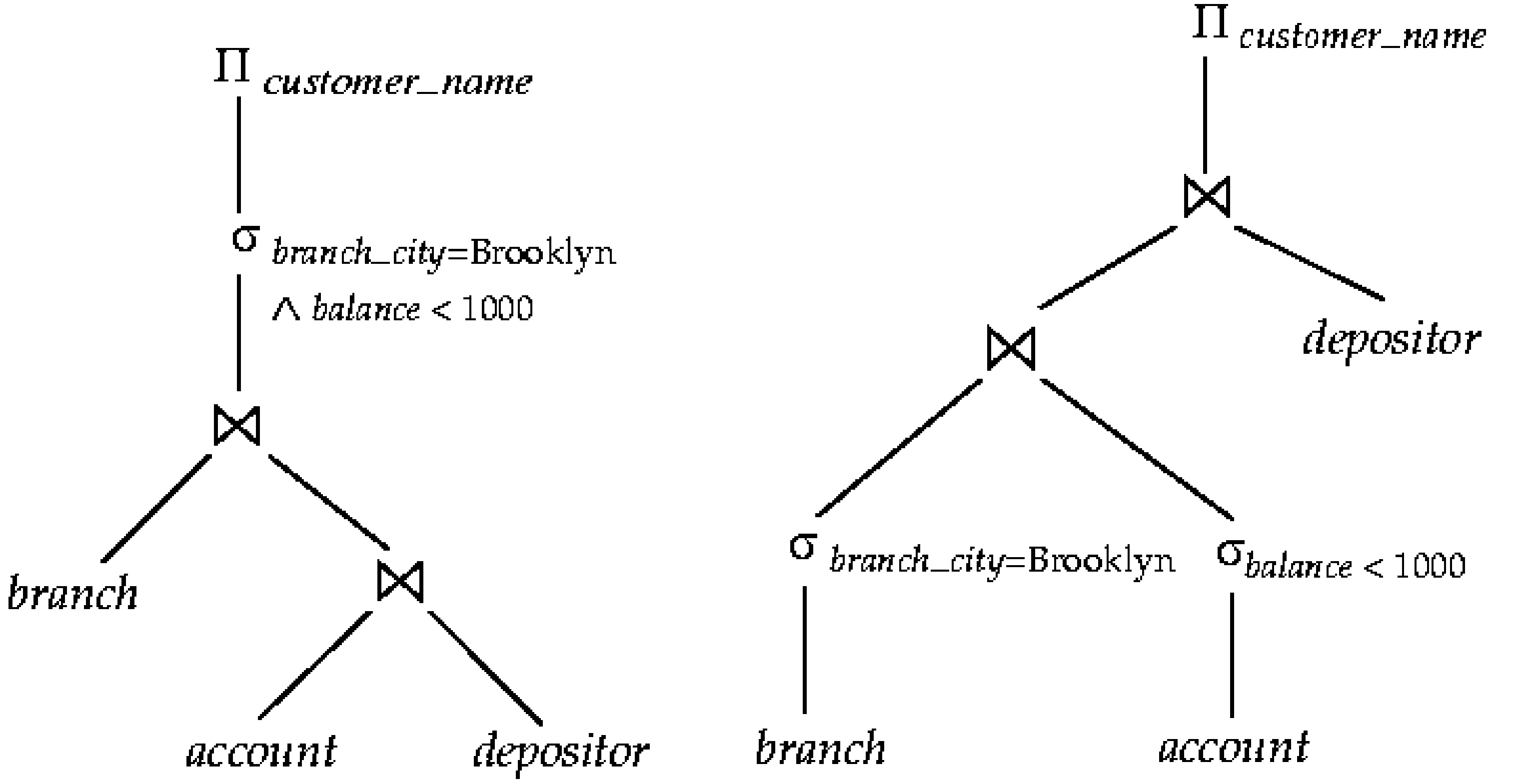
  - Based on rule 6a (join associativity):

$$\Pi_{customer\_name}((\sigma_{branch\_city = \text{“Brooklyn”} \wedge balance > 1000}(branch \bowtie account)) \bowtie depositor)$$

  - Now we can perform the selection earlier:

$$\sigma_{branch\_city = \text{“Brooklyn”}}(branch) \bowtie \sigma_{balance > 1000}(account)$$

Optimization
Pushing selection (example 2 illustrated)



(a) Initial expression tree (b) Tree after multiple transformations

Optimization
Pushing projection

- Example

$$\Pi_{customer\_name}((\sigma_{branch\_city\ =\ \text{“Brooklyn”}}\ (branch) \bowtie\ account) \bowtie depositor)$$

  - Eliminate the attributes no longer needed:

$$\Pi_{customer\_name}\ ((\Pi_{account\_number}\ \ (\sigma_{\text{branch\_city = “Brooklyn”}}\ (branch) \bowtie account\ ) \bowtie depositor\ )$$

- *By performing projection in advance, the size of the relations at join becomes smaller*

Optimization
Ordering at join

- According to rule 6:

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

- If $r_2 \bowtie r_3$ is larger than $r_1 \bowtie r_2$, than choose

$$(r_1 \bowtie r_2) \bowtie r_3$$

- Example

$$\Pi_{customer\_name}\ ((\sigma_{branch\_city = \text{“Brooklyn”}}\ (branch)) \bowtie (\text{account} \bowtie \text{depositor}))$$

Only a small number of clients have accounts at Brooklyn branch, therefore is more advantageous to execute first

$$\sigma_{branch\_city = \text{“Brooklyn”}}\ (branch) \bowtie account$$

- *For n relations there exist (2(n – 1))!/(n – 1)! different orderings for join.*
  - *n = 7 -> 665280, n = 10 ->176 bilions!*

To reduce the number of orderings under consideration, dynamic programming may be used

Cost estimation for logical plans

- l_r: dimension of a tuple in r *(in bytes)*.
- n_r: number of tuples in r.
- b_r: number of blocks used to store r.
- f_r: number of tuples in r that can be stored in a block

- If the tuples of r are stored in a single file (contiguous blocks on hard disk):

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

- *V(A, r):* number of distincst values of attribute A in r; equivalent to the dimension of $\prod_A(r)$ (on sets and not multi-sets).

- The logical plan generator estimates the number of tuples/blocks which result from each relational operator in the logical plan; these estimates are further used by the physical plan generator

Estimarea dimensiunii selecției

- $\sigma_{A=v}(r)$
  - n_r / $V(A,r)$: numărul de înregistrări ce satisfac selecția
  - pentru atribut cheie: 1
- $\sigma_{A \le V}(r)$ (cazul $\sigma_{A \ge V}(r)$ este simetric)
  - dacă sunt disponibile min(A,r) și max (A,r)
    - 0 dacă v < min(A,r)
    - $n_r \cdot \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$ altfel
  - dacă sunt disponibile histograme se poate rafina estimarea anterioară
  - în lipsa oricărei informații statistice dimensiunea se consideră a fi n_r / 2.

Estimarea dimensiunii selecțiilor complexe

- Selectivitatea unei condiții θ_i este probabilitatea ca un tuplu în relația r să satisfacă θ_i
  - dacă numărul de tuple ce satisfac θ_i este s_i , *selectivitatea* e s_i / n_r
- Conjuncția (în ipoteza independenței)

$\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(r)$:

$$n_r * \frac{s_1 * s_2 * \ldots * s_n}{n_r^n}$$

- Disjuncția

$\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(r)$:

$$n_r * \left(1 - (1 - \frac{s_1}{n_r}) * (1 - \frac{s_2}{n_r}) * \ldots * (1 - \frac{s_n}{n_r})\right)$$

- Negația

$\sigma_{\neg\theta}(r)$: $\quad n_r - size(\sigma_\theta(r))$

Estimarea dimensiunii joinului

- pentru produsul cartezian *r* x *s*: $n_r * n_s$ tuple, fiecare tuplu ocupă $s_r + s_s$ octeți
- pentru $r \bowtie s$
  - $R \cap S = \varnothing$: $n_r * n_s$
  - $R \cap S$ este o (super)cheie pentru R: <= n_s
  - $R \cap S = \{A\}$ nu e cheie pentru *R* sau *S*: $\frac{n_r * n_s}{V(A,s)}$ sau $\frac{n_r * n_s}{V(A,r)}$
    - minimul este considerat de acuratețe mai mare
    - dacă sunt disponibile histograme se calculează formulele anterioare pe fiecare celulă pentru cele două relații

Estimarea dimensiunii pentru alte operații

- Proiecția $\prod_A(r)$: $V(A,r)$
- Agregarea: $_A g_F(r)$: $V(A,r)$
- Operații pe mulțimi
  - $r \cup s$: $n_r + n_s$.
  - $r \cap s$: $\min(n_r, n_s)$
  - r-s : n_r
- Join extern
  - r ⟕ s: $dim(r \bowtie s) + n_r$
  - r ⟗ $s = dim(r \bowtie s) + n_r + n_s$
- $\sigma_{\theta 1}(r) \cap \sigma_{\theta 2}(r)$ echivalent cu $\sigma_{\theta 1}\sigma_{\theta 2}(r)$
- Estimatorii furnizează în general margini superioare

Physical plan optimization

Estimating costs for physical plans

- The cost is generally measured as the time needed to return the result
- Disk access is considered to be the most costly operation
  - Number of seeks * t_S (time to localize a single data block)
  - Number of blocks read/written * t_T (transfer time)
  - CPU cost is ignored for simplicity
- The cost for transferring b data blocks which required S seeks:

$$b * t_T + S * t_S$$

Algorithms for selection

- Linear search (full scan)
  - cost: $b_r * t_T + t_S$
  - if selection is over a key attribute, estimated cost: $b_r/2 * t_T + t_S$
  - may be applied for any search condition, data file ordering, existence of indexes
- Binary search
  - Applicable for equality conditions on the sort key
  - The cost of finding one qualifying tuple: $\lceil \log_2(b_r) \rceil * (t_T + t_S)$;

  If there exist several qualifying tuples only transfer time is added
- Index scan (suppose a B+-tree exists for the search key)
  - primary index on a candidate key, equality cond.: $(h_i + 1) * (t_T + t_S)$
  - primary index on a none-key, equality cond.: $h_i * (t_T + t_S) + t_S + t_T * b$
  - secondary index, equality, n tuples returned: $(h_i + n) * (t_T + t_S)$
  - primary index, range cond.: $h_i * (t_T + t_S) + t_S + t_T * b$
  - secondary index, range cond: ?

Algorithms for complex selections

- Conjunction: $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$
  - Use an index for θ_i and verify the rest when bringing data into memory
  - Use a multi-key index
  - Intersect the set of pointers returned by searching over all the indexes
- Disjunction: $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$
  - Union of the set of identifiers returned by index searches

Algorithms for join

- Algorithms:
  - nested-loop join
  - indexed nested-loop join
  - merge join
  - hash join

- Choosing from above implies cost estimation – requires estimates for the logical plan

Nested-loop joins

- For a theta-join: $r \bowtie_{\theta} s$:

  for each tuple t_r **in** r **do begin**

  for each tuple t_s **in** s **do begin**

  if (t_r, t_s) satisfies θ

  add $t_r \bullet t_s$ to the result set

  end

  end

- Inner relation – s
- External relation – r
- Estimated cost: $(n_r * b_s + b_r)*t_T + (n_r + b_r)*t_S$

- May be used for any join condition

Indexed nested-loop join

- Full file scans may be replaced by index scans if:
  - we deal with an **equi-join** (as a special case natural join)
  - there exists an index for the inner relation associated to the join attribute
- Idea: for every tuple t_r *in* r use the index to retrieve all the tuples in s satisfying the join condition - equivalent to a selection on s with the join condition
- Cost: $b_r\ (t_T + t_S) + n_r * c$
  - c is the cost of index search
  - if indexes for both relations are available, the relation with fewer tuples will be used as external within join
- Example:
  - *depositor* $\bowtie$ *customer, depositor* external relation
  - customer has a primary index of type B^+-tree on the join attribute *customer-name,* with m=20 entries per node
  - *customer:* 10,000 tuples (f=25), *depositors*:5000 tuples (f=50)
- cost: 100 + 5000 * 5 = 25,100 blocks transferred and seeks (compare to the case of standard nested-loop join: 2,000,100 blocks transferred and = 5100 seeks)

Merge join

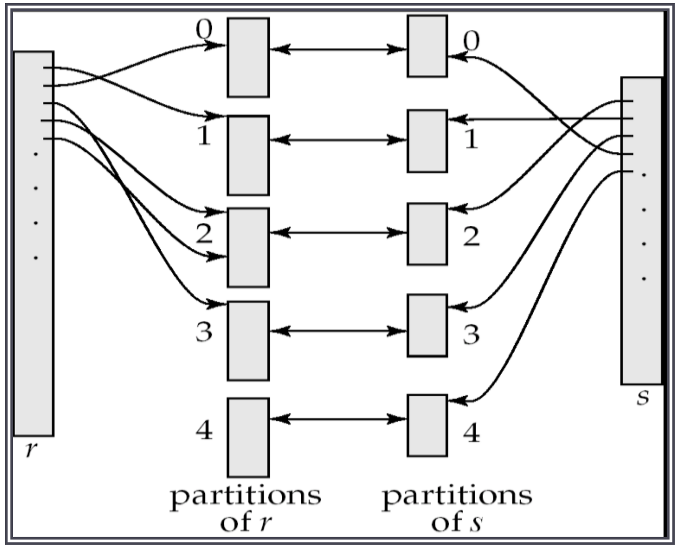
- May be used only for equi-joins

- Algorithm:
  1. Sort both relations based on the join attributes (luckily, they are stored ordered)
  2. Merge the two relations
- Cost:
  - $b_r + b_s$ transferred blocks
  - + the cost of sorting

- Hybrid merge join:
  - one relation is sorted, while for the second a B+ -tree associated to the join attribute is used
  - The sorted relation merges with the leaf level of the tree

Hash Join

- Applicable only for echi-join
- Algorithm: a hash function h aplied on the join attribute is used to partition the tuples of both relations into data blocks that fit in the main memory:
  - $r_1, r_2, \ldots r_n$
  - $s_{1,} s_2, \ldots s_n$
- tuples in r_i are compared only with tuples s_i



Complex joins

- Conjunction of conditions: $r \bowtie_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n} s$
  - Nested-loop join, verify all the conditions
  - Compute a simpler join $r \bowtie_{\theta i} s$ and afterwards verify the rest of conditions

- Disjunction of conditions : $r \bowtie_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n} s$
  - Nested-loop join, start verifying the conditions until one is satisfied
  - Compute the union of individual joins (applicable only for the set version of union)

$(r \bowtie_{\theta 1} s) \cup (r \bowtie_{\theta 2} s) \cup \ldots \cup (r \bowtie_{\theta n} s)$

Eliminating duplicates

- Based on sorting or hashing
- Because is costly, DBMSs eliminate tuples only when explicitly asked

Evaluating RA expressions (executing physical plans)

- The operators in the RA expression/tree are evaluated starting with the last level and moving up to the root

- Versions:
  - Materialize: (sub)expressions on lower levels are materialized as new relations (as data files stored on disks) and are given as entries for upper levels
  - Pipelining: tuples are given as entries to the operators on the upper levels when they are generated
      - Not always possible (think of hash join over merge join)
    - Consumer based: the upper level asks for new tuples
    - Producer based: the operator on the lower level writes in buffer and the parrent takes from the buffer (when the buffer is full there are waiting times on the lower level)

Inspecting execution plans in Oracle

- Record the plan:

EXPLAIN PLAN
    [SET STATEMENT_ID = <*id*>]
    [INTO <table_name>]
    FOR <sql_statement>;

  - Possible for any DML statement

- Visualizing the plan:

SELECT * FROM table(dbms_xplan.display);

or (not so nicely formatted)

select * from plan_table [where statement_id = <id>];

http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-explain-the-explain-plan-052011-393674.pdf

Execution plans in Oracle
Statistics

- Table statistics
  - Number of rows
  - Number of blocks
  - Average row length
- Column statistics
  - Number of distinct values (NDV) in column
  - Number of nulls in column
  - Data distribution (histogram)
- Index statistics
  - Number of leaf blocks
  - Levels
  - Clustering factor
- System statistics
  - I/O performance and utilization
  - CPU performance and utilization

Execution plans in Oracle
Collecting statistics

- Procedures in package DBMS_STATS:

- GATHER_INDEX_STATS
  - Index statistics
- GATHER_TABLE_STATS
  - Table, column, and index statistics
- GATHER_SCHEMA_STATS
  - Statistics for all objects in a schema
- GATHER_DATABASE_STATS
  - Statistics for all objects in a database
- GATHER_SYSTEM_STATS
  - CPU and I/O statistics for the system

- http://docs.oracle.com/cd/B10500_01/server.920/a96533/stats.htm

Execution plans in Oracle
Hints

- When launching a query it is possible to indicate the Oracle optimizer some choices for the execution plan:

```
SELECT /*+ USE_MERGE(employees departments) */ * FROM employees, departments WHERE employees.department_id =
    departments.department_id;
```

http://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements006.htm

References

- Chapters13 and 14 in *Avi Silberschatz Henry F. Korth S. Sudarshan. "Database System Concepts"*. McGraw-Hill Science/Engineering/Math; 4th edition