

# Databases

## Lecture 11

Query Optimization in Relational Databases

Evaluating Relational Algebra Operators

# SQL Statements Execution

- client application - SQL statement execution request
  - for any query - minimum response time
- statement execution - stages:
  - client: generate SQL statement (non-procedural language), send it to server
  - server:
    - analyze SQL statement (syntactically)
    - translate statement into an internal form (relational algebra expression)
    - transform internal form into an optimal form
    - generate a procedural execution plan
    - evaluate procedural plan, send result to client

- the following operators are necessary in the querying process:
  - selection:  $\sigma_C(R)$
  - projection:  $\pi_\alpha(R)$
  - cross-product:  $R_1 \times R_2$
  - union:  $R_1 \cup R_2$
  - set-difference:  $R_1 - R_2$
  - intersection:  $R_1 \cap R_2$
  - theta join:  $R_1 \otimes_\Theta R_2$
  - natural join:  $R_1 * R_2$
  - left outer join:  $R_1 \bowtie_C R_2$
  - right outer join:  $R_1 \bowtie_C R_2$
  - full outer join:  $R_1 \bowtie_C R_2$
  - left semi join:  $R_1 \triangleright R_2$
  - right semi join:  $R_1 \triangleleft R_2$
  - division:  $R_1 \div R_2$
  - duplicate elimination:  $\delta(R)$
  - sorting:  $S_{\{list\}}(R)$
  - grouping:  $\gamma_{\{list1\} \text{ group by } \{list2\}}(R)$

- an SQL query can be written in multiple ways
- example for a relational database
- primary keys are underlined, foreign keys are written in blue  
programs[id, pname, pdescription]  
groups[id, **program**, yearofstudy, gdescription]  
students[cnp, lastname, firstname, **sgroup**, gpa, addr, email]
- query: find students (lastname, firstname, year of study, program name, gpa) in a given program (e.g., with id = 2, can be a parameter), with a gpa >= 9 (can be a parameter):

a)

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM students st, groups gr, programs pr
WHERE st.sgroup = gr.id AND gr.program = pr.id
      AND program = 2 and gpa >= 9
```

b)

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM (students st INNER JOIN groups gr ON
      st.sgroup = gr.id)
      INNER JOIN programs pr ON gr.program = pr.id
WHERE program = 2 AND gpa >= 9
```

c)

```
SELECT lastname, firstname, yearofstudy, pname, gpa  
FROM
```

```
(
```

```
  (SELECT lastname, firstname, sgroup, gpa  
    FROM students
```

```
    WHERE gpa >= 9) st
```

```
INNER JOIN
```

```
  (SELECT * FROM groups WHERE program = 2) gr  
    ON st.sgroup = gr.id
```

```
)
```

```
INNER JOIN
```

```
  (SELECT id, pname FROM programs WHERE id = 2) pr
```

```
ON gr.program = pr.id
```

- the previous query versions are equivalent (they provide the same answer)
- equivalent relational algebra expressions:

a.

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM students st, groups gr, programs pr
WHERE st.sgroup = gr.id AND gr.program = pr.id
      AND program = 2 and gpa >= 9
```

$$\pi_{\beta}(\sigma_C(students \times groups \times programs))$$



b)

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM (students st INNER JOIN groups gr ON
      st.sgroup = gr.id)
      INNER JOIN programs pr ON gr.program = pr.id
WHERE program = 2 AND gpa >= 9
```

$$\pi_{\beta}(\sigma_{c_1}((students \otimes_{c_2} groups) \otimes_{c_3} programs))$$

c)

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM
  (
    (SELECT lastname, firstname, sgroup, gpa
     FROM students
     WHERE gpa >= 9) st
    INNER JOIN
      (SELECT * FROM groups WHERE program = 2) gr
      ON st.sgroup = gr.id
  )
  INNER JOIN
    (SELECT id, pname FROM programs WHERE id = 2) pr
    ON gr.program = pr.id
```

$$\pi_{\beta}(((\pi_{\beta_1}(\sigma_{c_2}(students))) \otimes_{c_3} (\sigma_{c_4}(groups))) \otimes_{c_5} (\pi_{\beta_2}(\sigma_{c_6}(programs))))$$

- an evaluation tree can be constructed for a relational algebra expression
- problems:
  - which version is better?
  - when generating the execution plan:
    - which parameters are optimized?
    - what information is required?
  - what can the optimizer (DBMS component) do?

# Relational Algebra Operators - Evaluation

- operands for relational operators:
  - database tables (can have attached indexes)
  - temporary tables (obtained by evaluating some relational operators)
- several evaluation algorithms could be used for a relational algebra operator
- when generating the execution plan:
  - choose the algorithm with the lowest complexity (for the current database context); take into account data from the system catalog, statistical information

## \* algorithms

### Table Scan

- many operators require a full scan of the entire table
- $b_R$  - number of blocks storing a table's records
  - sequential search algorithm - approximately  $b_R/2$  blocks are necessary (on average) when performing a sequential search on a key value
  - all blocks must be brought into main memory when performing a sequential search on a non-key field
- high transfer time for large tables

## Index Seek

- searching for a key value  $K_0$
- condition of the form:  $K = K_0$
- search:
  - explicit (searching a table, evaluating a join)
  - implicit (checking a key constraint)
- examine an index (stored as a B-tree, B+ tree) created:
  - via a key constraint
  - with the CREATE INDEX statement
- obs:  $K$  can be a simple or composite key

## Index Scan

- evaluating  $\sigma_C(R)$ , where condition  $C$  is of the form:
  - $A < v, A \leq v, A > v, A \geq v, A \text{ IS NULL}, A \text{ IS NOT NULL}$  – index built for a key  $A$
  - $A = v, A < v, A \leq v, A > v, A \geq v, A \text{ IS NULL}, A \text{ IS NOT NULL}$  – index built for a non-key field  $A$
- partial / total index scan – obtain desired records' addresses
- get records from the relation; some blocks can be read multiple times



- a join can be defined as a cross-product followed by a selection
- joins arise more often in practice than cross-products
- in general, the result of a cross-product is much larger than the result of a join
- it's important to implement the join without materializing the underlying cross-product, by applying selections and projections as soon as possible, and materializing only the subset of the cross-product that will appear in the result of the join

## Cross Join

- this algorithm is used to evaluate a cross-product:
  - R CROSS JOIN S
  - R INNER JOIN S ON C (C evaluates to TRUE)
  - SELECT ... FROM R, S ..., no join condition between R and S
- $b_R, b_S$ 
  - the number of blocks storing R and S, respectively
- m, n
  - the number of blocks from R and S that can simultaneously appear in the main memory (there are m+n buffers for the 2 tables)

## Cross Join

- the following algorithm can be used to generate the cross-product  $\{(r, s) \mid r \in R, s \in S\}$ :
- for every group of max.  $m$  blocks in  $R$ :
  - read the group of blocks from  $R$  into main memory; let  $M_1$  be the set of records in these blocks
  - for every group of max.  $n$  blocks in  $S$ :
    - read the group of blocks from  $S$  into main memory; let  $M_2$  be the set of records in these blocks
    - for every  $r \in M_1$ :
      - for every  $s \in M_2$ : add  $(r, s)$  to the result

## Cross Join

- algorithm complexity: total number of read blocks (from the 2 tables):

$$b_R + \left\lceil \frac{b_R}{m} \right\rceil * b_S \quad (1)$$

(number of blocks in R; for every group of max. m blocks in R, read S)

- to minimize this value, m should be maximized (the other operands are constants); one buffer can be used for S (so n = 1), while the remaining space can be used for R (m max.)
- switch the 2 relations (in the algorithm and when computing the complexity)  
=> complexity:

$$b_S + \left\lceil \frac{b_S}{n} \right\rceil * b_R \quad (2)$$

- choose better version
- obs.: if  $b_R \leq m$  or  $b_S \leq n \Rightarrow$  complexity  $b_R + b_S$

## Nested Loops Join

- the Cross Join algorithm can be used to evaluate a join between 2 tables
- for every element  $(r, s)$  in the cross-product, evaluate the condition in the join operator
- elements  $(r, s)$  that don't meet the join condition are eliminated

## Indexed Nested Loops Join

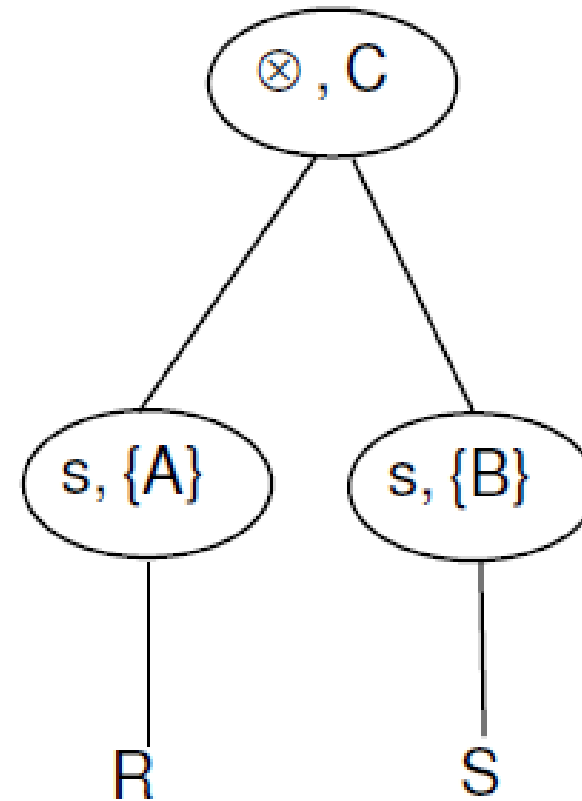
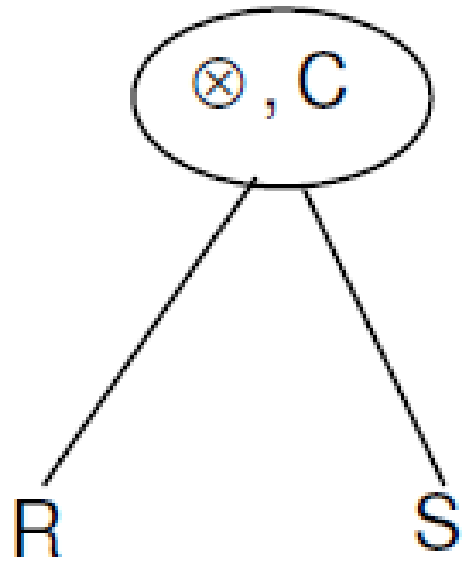
- this algorithm is used to evaluate  $R \bowtie_C S$ , where  $C \equiv (R.A=S.B)$ , and there is an index on A (in R) or on B (in S)
- in the algorithm description below, we assume there is an index on column B in table S
- for every block in R:
  - read the block into main memory; let M be the set of records in the block
  - for every  $r \in M$ :
    - determine  $v = \pi_A(r)$
    - use the index on B in S to determine records  $s \in S$  with value v for B; for every such record s, the pair (r,s) is added to the result
- obs.: depending on the type of index - at most 1 / multiple matching records in S

## Merge Join

- this algorithm is used to evaluate  $R \bowtie_C S$ , where  $C \equiv (R.A=S.B)$ , and there are no indexes on A (in R) and B (in S)
- sort R and S on the columns used in the join: R on A, S on B
- scan obtained tables; let r in R and s in S be 2 current records
  - if  $r.A = s.B$ : add  $(r', s')$  to the result;  $r'$  is in the set of all consecutive records in R with  $A = r.A$ , analogous for  $s'$  and S;  $\text{next}(r)$ ;  $\text{next}(s)$  (get a record with the next value for A and B)
  - if  $r.A < s.B$ :  $\text{next}(r)$  (determine record in sorted R with the next value for A)
  - if  $r.A > s.B$ :  $\text{next}(s)$  (determine record in sorted S with the next value for B)

## Merge Join

- this algorithm replaces an evaluation tree with another evaluation tree:





## Hash Join

- this algorithm is used to evaluate  $R \bowtie_C S$ , where  $C \equiv (R.A = S.B)$

### 1. partitioning phase

- hash R and S on the join column, use the same hash function  $h$

=> partitions

### 2. probing phase

- tuples in partition  $R_x$  are compared only with tuples in partition  $S_x$  (tuples in partition  $R_1$  cannot join with tuples in partition  $S_2$ , for instance, as they have a different hash value)

## Outer Joins

- adapt condition join algorithms

## Operations on Sets of Records: $R \cup S$ , $R - S$ , $R \cap S$

- adapt previous algorithms
- e.g., intersection:
  - sort  $R$  using all columns, sort  $S$  using all columns
  - scan sorted  $R$  and  $S$ , write in the result only the tuples in  $R$  that also appear in  $S$

# Relational Algebra Equivalences

- SQL statement - transformed into a relational algebra expression (based on a set of transformation rules for the clauses that appear in the statement)
- transform relational expression (such that the evaluation algorithm has a lower complexity)
- certain transformation rules are used (mathematical properties of the relational operators)

$$* \sigma_C(\pi_\alpha(R)) = \pi_\alpha(\sigma_C(R))$$

- selection reduces the number of records for projection; in the second expression, the projection operator analyzes fewer records
- optimization - algorithm that evaluates both operators in a single pass of R

\* perform one pass instead of 2:

$$\sigma_{C_1}(\sigma_{C_2}(R)) = \sigma_{C_1 \text{ AND } C_2}(R)$$

\* replace cross-product and selection by condition join (a number of condition join algorithms don't evaluate the cross-product):

$$\sigma_C(R \times S) = R \bowtie_C S$$

, where C - join condition between R and S

\* R and S - compatible schemas:

$$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$$

$$\sigma_C(R \cap S) = \sigma_C(R) \cap \sigma_C(S)$$

$$\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S)$$

\*  $\sigma_C(R \times S)$

particular cases:

- C contains only attributes from R:

$$\sigma_C(R \times S) = \sigma_C(R) \times S$$

- $C = C1 \text{ AND } C2$ ,  $C1$  contains only attributes from R,  $C2$  - only attributes from S:

$$\sigma_{C1 \text{ AND } C2}(R \times S) = \sigma_{C1}(R) \times \sigma_{C2}(S)$$

- $C = C1 \text{ AND } C2$ ,  $C2$  - join condition between R and S:

$$\sigma_{C1 \text{ AND } C2}(R \times S) = \sigma_{C1}(R \bowtie_{C2} S)$$

\*  $\pi_{\alpha}(R \cup S) = \pi_{\alpha}(R) \cup \pi_{\alpha}(S)$

\*  $\pi_{\alpha}(R \otimes_C S) = \pi_{\alpha}(\pi_{\alpha_1}(R) \otimes_C \pi_{\alpha_2}(S))$

- $\alpha_1$ : attributes in R that appear in  $\alpha$  or C
- $\alpha_2$ : attributes in S that appear in  $\alpha$  or C

\* associativity and commutativity for some relational operators

- associativity and commutativity for  $\cup$  and  $\cap$
- associativity for the cross-product and the natural join
- "equivalent" results (same records, but different column order) when commuting operands in  $\times$  and certain join operators
  - $R \times S = S \times R$  – when using the Cross Join algorithm, the order of the data sources is important

\* transitivity of some relational operators for the join operators - additional filters could be applied before the join:

- $(A > B \text{ AND } B > 3) \equiv (A > B \text{ AND } B > 3 \text{ AND } A > 3)$

- example: A is in R, B is in S:

$$R \otimes_{A > B \text{ AND } B > 3} S = (\sigma_{A > 3}(R)) \otimes_{A > B} (\sigma_{B > 3}(S))$$

- $(A = B \text{ AND } B = 3) \equiv (A = B \text{ AND } B = 3 \text{ AND } A = 3)$

- example: A is in R, B is in S:

$$R \otimes_{A = B \text{ AND } B = 3} S = (\sigma_{A = 3}(R)) \otimes_{A = B} (\sigma_{B = 3}(S))$$

\* evaluating  $\sigma_C(R)$ , where  $C \equiv (R.A \in \delta(\pi_{\{B\}}(S)))$ ; avoid evaluating C for every record of R; the initial evaluation is equivalent to:

$$R \otimes_{R.A = S.B} (\delta(\pi_{\{B\}}(S)))$$



- consider again the query described on the database:  
programs[id, pname, pdescription]  
groups[id, program, yearofstudy, gdescription]  
students[cnp, lastname, firstname, sgroup, gpa, addr, email]
- query: find students (lastname, firstname, year of study, program name, gpa) in a given program (e.g., with id = 2), with a gpa  $\geq 9$ :

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM students, groups, programs
WHERE students.sgroup = groups.id AND
      groups.program = programs.id AND
      program = 2 and gpa  $\geq$  9
```

- denote by:

$C \equiv (\text{students.sgroup} = \text{groups.id} \text{ AND } \text{groups.program} = \text{programs.id} \text{ AND } \text{program} = 2 \text{ and } \text{gpa} \geq 9)$

$\beta = \{\text{lastname}, \text{firstname}, \text{yearofstudy}, \text{pname}, \text{gpa}\}$  – attributes in the SELECT clause

- the corresponding relational expression:

$$\pi_{\beta}(\sigma_C(\text{students} \times \text{groups} \times \text{programs}))$$

- carry out the following transformations, using previously discussed rules:
- associativity for  $\times$ :
 
$$students \times groups \times programs = (students \times groups) \times programs \quad \text{or}$$

$$students \times groups \times programs = students \times (groups \times programs)$$
- commute  $\sigma$  with  $\times$  (use a particular case); the transitivity of the equality operator:

(groups.program = programs.id AND program = 2)

$\equiv$  (groups.program = programs.id AND program = 2 AND **programs.id = 2**)

students.sgroup = groups.id AND groups.program = programs.id AND program = 2 AND gpa >= 9 <b>AND programs.id = 2</b>				
C1	C2	C3	C4	C5

$\sigma_C(students \times groups \times programs) =$   
 $\sigma_{C1 \text{ AND } C2}((\sigma_{C4}(students) \times \sigma_{C3}(groups)) \times \sigma_{C5}(programs)) \text{ or}$   
 $\sigma_{C1 \text{ AND } C2}(\sigma_{C4}(students) \times (\sigma_{C3}(groups) \times \sigma_{C5}(programs)))$

- replace selection and cross-product with condition join:

$$= ((\sigma_{c_4}(students)) \otimes_{c_1} (\sigma_{c_3}(groups))) \otimes_{c_2} (\sigma_{c_5}(programs))$$

or

$$= (\sigma_{c_4}(students)) \otimes_{c_1} ((\sigma_{c_3}(groups)) \otimes_{c_2} (\sigma_{c_5}(programs)))$$

- choose a version based on statistical information from the database; we consider the first version:

$$\Rightarrow e = \pi_{\beta}(((\sigma_{c_4}(students)) \otimes_{c_1} (\sigma_{c_3}(groups))) \otimes_{c_2} (\sigma_{c_5}(programs)))$$

- commute  $\pi$  with join:

$\beta 1 = \{\text{lastname, firstname, gpa, sgroup}\}$  - useful for  $\beta$  and join

$\beta 2 = \{\text{id, program, yearofstudy}\}$  - useful for  $\beta$  and join

$\beta 3 = \{\text{id, pname}\}$  - useful for  $\beta$  and join

$$e = \pi_{\beta}(((\pi_{\beta 1}(\sigma_{c_4}(students))) \otimes_{c_1} (\pi_{\beta 2}(\sigma_{c_3}(groups)))) \otimes_{c_2} (\pi_{\beta 3}(\sigma_{c_5}(programs))))$$

- the last expression corresponds to the statement:

```
SELECT lastname, firstname, yearofstudy, pname, gpa
```

```
FROM
```

```
(
```

```
(SELECT lastname, firstname, gpa, sgroup FROM students WHERE gpa >= 9) st
```

```
INNER JOIN
```

```
(SELECT id, program, yearofstudy FROM groups WHERE program = 2) gr
```

```
ON st.sgroup = gr.id
```

```
)
```

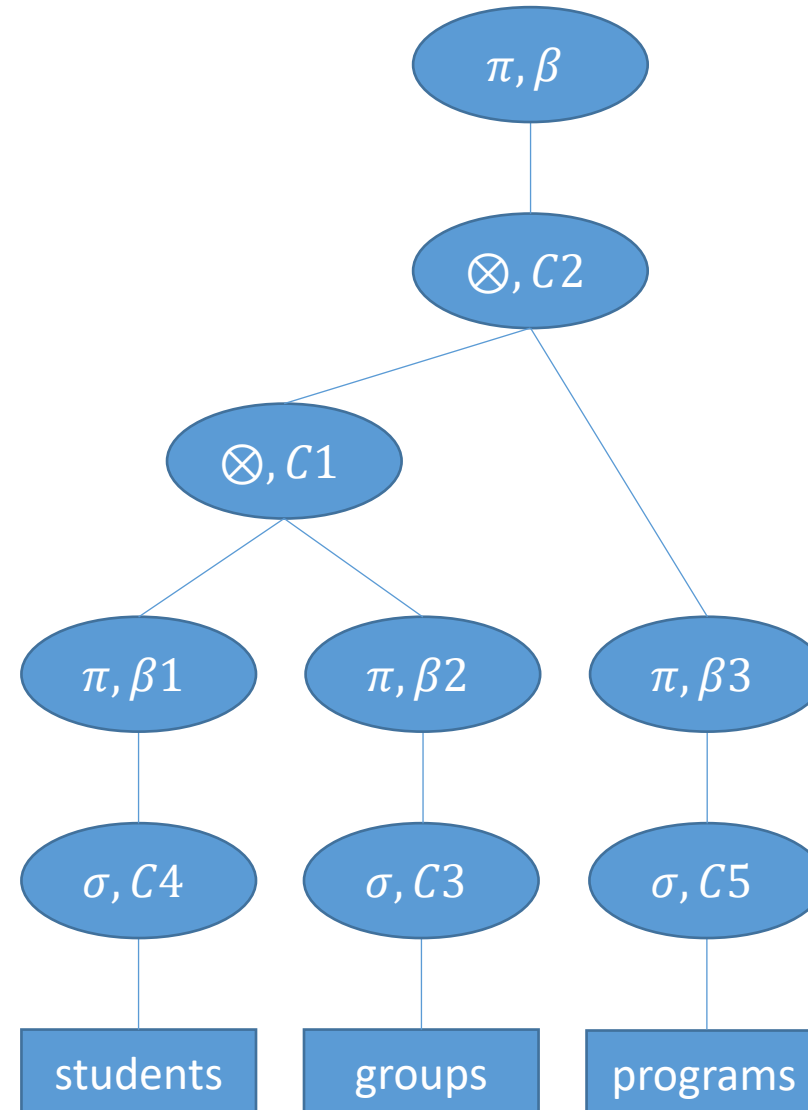
```
INNER JOIN
```

```
(SELECT id, pname FROM programs WHERE programs.id = 2) pr
```

```
ON gr.program = pr.id
```

- an evaluation tree can be constructed for the last version of the relational algebra expression

- using information from the system catalog and possibly statistical information, an execution plan can be generated from the last version of the expression; every relational operator is replaced by an evaluation algorithm



## References

- [Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Si11] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts (6th Edition), McGraw-Hill, 2011
- [Kn76] KNUTH, D.E., Tratat de programare a calculatoarelor. Sortare și căutare. Ed. Tehnică, București, 1976
- [Ga09] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book (2nd Edition), Pearson Education, 2009