Databases

Lecture 11

Query Optimization in Relational Databases

Evaluating Relational Algebra Operators

SQL statement execution

- client application executes SQL statement
    - for any query - minimum response time
    - when creating the SQL statement, the physical structure of the database is not necessarily known
- statement execution - stages:
    - client: generate SQL statement (non-procedural language), send it to server
    - server:
        - analyze SQL statement - syntactically / semantically
        - translate statement into an internal form (relational algebra expression)
            - replace existing views with corresponding expressions
        - transform internal form into an optimal form
        - generate a procedural execution plan ->

    - evaluate procedural plan, send result to client

- analyze generated execution plans:
  - periodically
  - after changing the physical structure
  - after performing multiple change operations on tables
- such analyses can lead to further optimizations (e.g., changing the physical structure)
- relational algebra operators are used when generating the execution plan

- 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: $R_1 \ltimes_C R_2$
  - right outer join: $R_1 \rtimes_C R_2$
  - full outer join: $R_1 \bowtie_C R_2$
  - left semi join: $R_1 \rhd R_2$
  - right semi join: $R_1 \lhd R_2$
  - division: $R_1 \div R_2$
  - duplicate elimination: $\delta(R)$
  - sorting: $S_{\{list\}}(R)$
  - grouping: $\gamma_{\{\text{list1}\} \text{ group by } \{\text{list2}\}}(R)$

- a 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]

  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
```

$$\Pi_{\beta}(\sigma_{C1}((students \otimes_{C2} groups) \otimes_{C3} programs))$$

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
```

$$\pi_{\beta}(((\pi_{\beta 1}(\sigma_{c2}(students))) \otimes_{C3} (\sigma_{C4}(groups))) \otimes_{C5} (\pi_{\beta 2}(\sigma_{C6}(programs))))$$

- 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?

evaluating relational algebra operators - algorithms

- operands for relational operators:
  - database tables (can have attached indexes)
  - views or temporary tables (obtained by evaluating some relational operators) without indexes
- 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

A1. 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

A2. Index Seek

- the most performant algorithm
- used when searching for a key value K_0
- condition of the form: $K = K_0$
- search:
  - explicit, e.g., searching a table, evaluating a join
  - implicit, e.g., 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

A3. Index Scan

- algorithm used to evaluate operator $\sigma_c(R)$, where condition C is of the form:
    - A < v, A <= v, A > v, A >= v, A IS NULL, A IS NOT NULL – index built for a key A
    - A = v, A < v, A <= v, A > v, A >= v, A IS NULL, A 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

A3. Index Scan

- obs:

1. this algorithm's execution time can be quite significant; the Table Scan algorithm could be faster

- if the number of records in the answer can be estimated, one can deduce which of the 2 algorithms is better
- an *evaluation history* could retain the number of records obtained at previous evaluations of the same search

2. if such an estimation is not possible, the client can propose a certain evaluation version via the character string in the SELECT statement

- 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

A4. Cross Join

- this algorithm is used to evaluate a cross product:
    - R CROSS JOIN S
    - R INNER JOIN S ON TRUE
    - SELECT ... FROM R, S ..., no link 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, i.e., there are m+n buffer slots for the 2 tables

A4. 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
```

A4. Cross Join

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

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

(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 slot 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:

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

- choose better version

A4. Cross Join

- m=n => choose smaller table as the first table (min{ b_R, b_S })
- obs.: if $b_R <= m$ or $b_S <= n$ => complexity $b_R + b_S$

A5. 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

A6. Indexed Nested Loops Join

- this algorithm is used to evaluate $R \otimes_C S$, where C ≡ (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 ∈ M:
        • determine v = $\Pi_A(r)$
        • use the index on B in S to determine records s ∈ S with
          value v for B; for every such record s, the pair (r,s)
          is added to the result
```

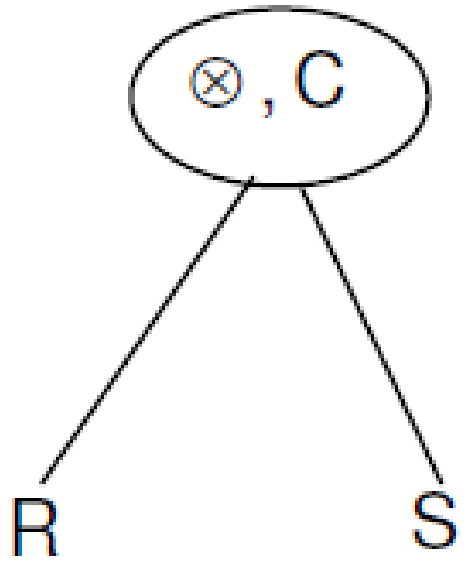
- obs.: depending on the type of index - at most 1 / multiple matching records in S

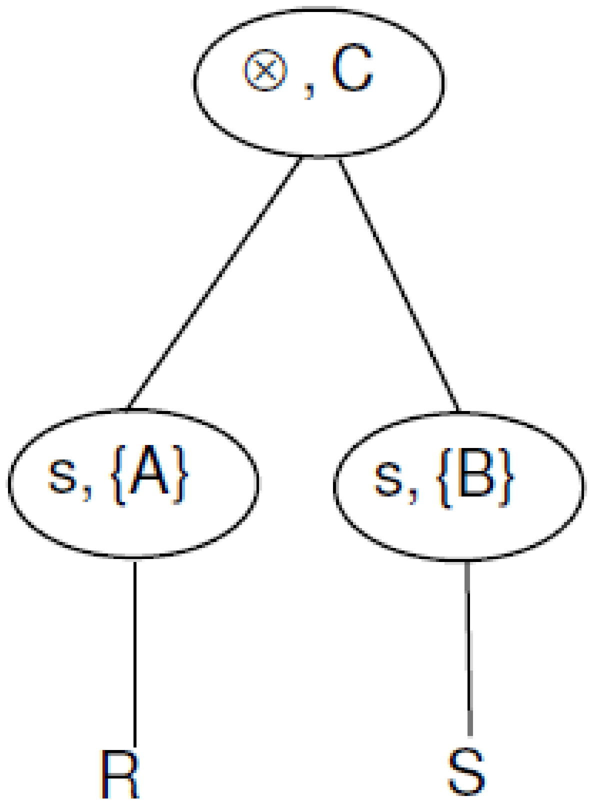
A7. Merge Join

- this algorithm is used to evaluate $R \otimes_C S$, where C ≡ (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 in parallel; 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; next(r); next(s) (get a record with the next value for A and B)
  - if r.A < s.B: next(r) (determine record in sorted R with the next value for A)
  - if r.A > s.B: next(s) (determine record in sorted S with the next value for B)

A7. Merge Join

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





A8. Hybrid Merge Join

- this algorithm is used to evaluate $\mathrm{R} \otimes_{\mathrm{C}} \mathrm{S}$, where:
    - C ≡ (R.A = S.B)
    - for one of the join attributes (A in R or B in S) the source is already sorted or it is sorted for the algorithm
    - for the other attribute there is a B+ tree index
- merge sorted source and terminals in B+ tree index (created for the second table)

A9. Hash Join

- this algorithm is used to evaluate $\mathrm{R} \otimes_{\mathrm{C}} \mathrm{S}$, where C ≡ (R.A = S.B)

1. store one of the tables (preferably the smaller one) in the main memory or in temporary files, using a hashing function h; assume S is this table

2. for every r ∈ R, determine v = h(r.A); compare record r only with records in S that are in the same *partition*

- obs: the algorithm is very efficient if one of the tables can be stored in main memory; good results can also be obtained for large tables outer joins

- adapt condition join algorithms

multiple joins

- can use the associativity / commutativity properties
- example - natural join for 4 tables using only associativity:
  - ((R1 * R2) * R3) * R4
  - (R1 * R2) * (R3 * R4)
  - (R1 * (R2 * (R3 * R4)))
  - (R1 * (R2 * R3)) * R4
  - R1 * ((R2 * R3) * R4)
- associate an evaluation tree with each expression; inner nodes correspond to intermediate results; choose execution plan with minimum *estimated evaluation time*

operations on sets of records: R ∪ S, R – S, R ∩ S

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

relational algebra expression - optimal form

- the SQL statement is transformed into a relational algebra expression (based on a set of transformation rules for the clauses that appear in the statement)
- without additional information from the database, the relational expression can be transformed into an optimal form (the evaluation algorithm has a lower complexity)
- certain transformation rules are used (mathematical properties of the relational operators)
- every DBMS uses certain transformation rules

R1. $\sigma_C\left(\Pi_\alpha(R)\right) = \Pi_\alpha\left(\sigma_C(R)\right)$

- 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

R2. perform one pass instead of 2:

$$\sigma_{C1}\left(\sigma_{C2}(R)\right) = \sigma_{C1\ \text{AND}\ C2}(R)$$

$$\Pi_\alpha\left(\Pi_\beta(R)\right) = \Pi_{\alpha\cap\beta}(R)$$

R3. replace cross product and selection by condition join (a number of condition join algorithms don't evaluate the cross product):

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

, where C - link condition between R and S

R4. use distributivity properties (of a unary operator "f" with respect to a binary operator "$\odot$"):

$$f(a \odot b) = f(a) \odot f(b)$$

R4.a. σ is distributive wrt ∪**,** ∩**,** − (R and S must have compatible schemas):

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

R4.b. σ is distributive wrt ×

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

particular cases:

- C contains only attributes from R:

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

R4.b. σ is distributive wrt ×

particular cases:

- C = C1 AND C2, C1 contains only attributes from R, C2 - only attributes from S:

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

- C = C1 AND C2, C2 - link condition between R and S:

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

R4.c. Π is distributive wrt ∪

$$\Pi_{\alpha}(R \cup S) = \Pi_{\alpha}(R) \cup \Pi_{\alpha}(S)$$

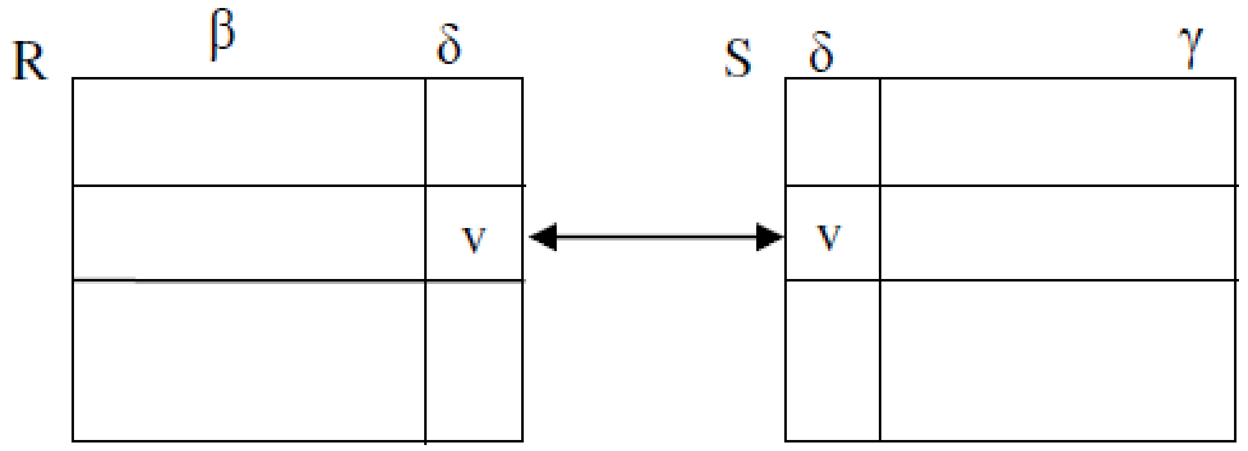
- obs: Π is not distributive wrt ∩, −
- example for set-difference:

$$\Pi_{\{x\}}\left(\begin{array}{cc} x & y \\ \left[\begin{array}{cc} 1 & 2 \\ 1 & 3 \\ 2 & 3 \end{array}\right] \end{array} - \begin{array}{cc} x & y \\ \left\{\begin{array}{cc} 2 & 3 \\ 1 & 4 \end{array}\right\} \end{array} \right)$$

R4.d. Π is distributive wrt "join“:

$$\Pi_{\alpha}(\mathrm{R}\ \mathrm{join}\ \mathrm{S}) = \Pi_{\alpha}(\Pi_{\alpha 1}(\mathrm{R})\ \mathrm{join}\ \Pi_{\alpha 2}(\mathrm{S})),$$ where

- $\alpha 1$ = ((attributes in α) ∩ (attributes in R)) ∪ (attributes for the join)
- $\alpha 2$ = ((attributes in α) ∩ (attributes in S)) ∪ (attributes for the join)



$$\alpha \subset \beta \cup \gamma \cup \delta$$
$$\alpha 1 = (\alpha \cap \beta) \cup \delta$$
$$\alpha 2 = (\alpha \cap \gamma) \cup \delta$$

R5. associativity and commutativity for some relational operators:

R5. a. associativity and commutativity for ∪ and ∩

R5. b. associativity for the cross product and the natural join

R5. c. "equivalent" results (same records, but different column order) when commuting operands in × and certain join operators

- R × S = S × R – when using the Cross Join algorithm, the order of the data sources is important
- R join S = S join R – when using Hash Join, the hash function should be used on the smaller source

R6. transitivity of some relational operators for the join operators - additional filters could be applied before the join:

- (A>B AND B>3) ≡ (A>B AND B>3 AND A>3)
- example: A is in R, B is in S:

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

- (A=B AND B=3) ≡ (A=B AND B=3 AND A=3)
- example: A is in R, B is in S:

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

R7. 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]

  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 >= 9:

```
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
```

- denote by:

C $\equiv$ (st.sgroup = gr.id AND gr.program = pr.id AND program = 2 and gpa >= 9)

β = {lastname, firstname, yearofstudy, pname, gpa} – attributes in the SELECT clause

- the corresponding relational expression:

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

- perform the following transformations, using previously discussed rules:
  - associativity for ×:

$$students \times groups \times programs = (students \times groups) \times programs$$ or

$$students \times groups \times programs = students \times (groups \times programs)$$

- distributivity of σ wrt ×, using a particular case, and the transitivity of the equality operator:

(gr.program = pr.id AND program = 2)

≡ (gr.program = pr.id AND program = 2 AND programs.id = 2)

st.sgroup = gr.id	AND	gr.program = pr.id	AND	program = 2	AND	gpa >= 9	AND	programs.id = 2
C1		C2		C3		C4		C5

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

- replace selection and cross product with condition join:

$$= ((\sigma_{C4}(students)) \otimes_{C1} (\sigma_{C3}(groups))) \otimes_{C2} (\sigma_{C5}(programs))$$

or

$$= (\sigma_{C4}(students)) \otimes_{C1} ((\sigma_{C3}(groups)) \otimes_{C2} (\sigma_{C5}(programs)))$$

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

$$\Rightarrow e = \pi_{\beta}(((\sigma_{C4}(students)) \otimes_{C1} (\sigma_{C3}(groups))) \otimes_{C2} (\sigma_{C5}(programs)))$$

- distributivity of Π wrt join:

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

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

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

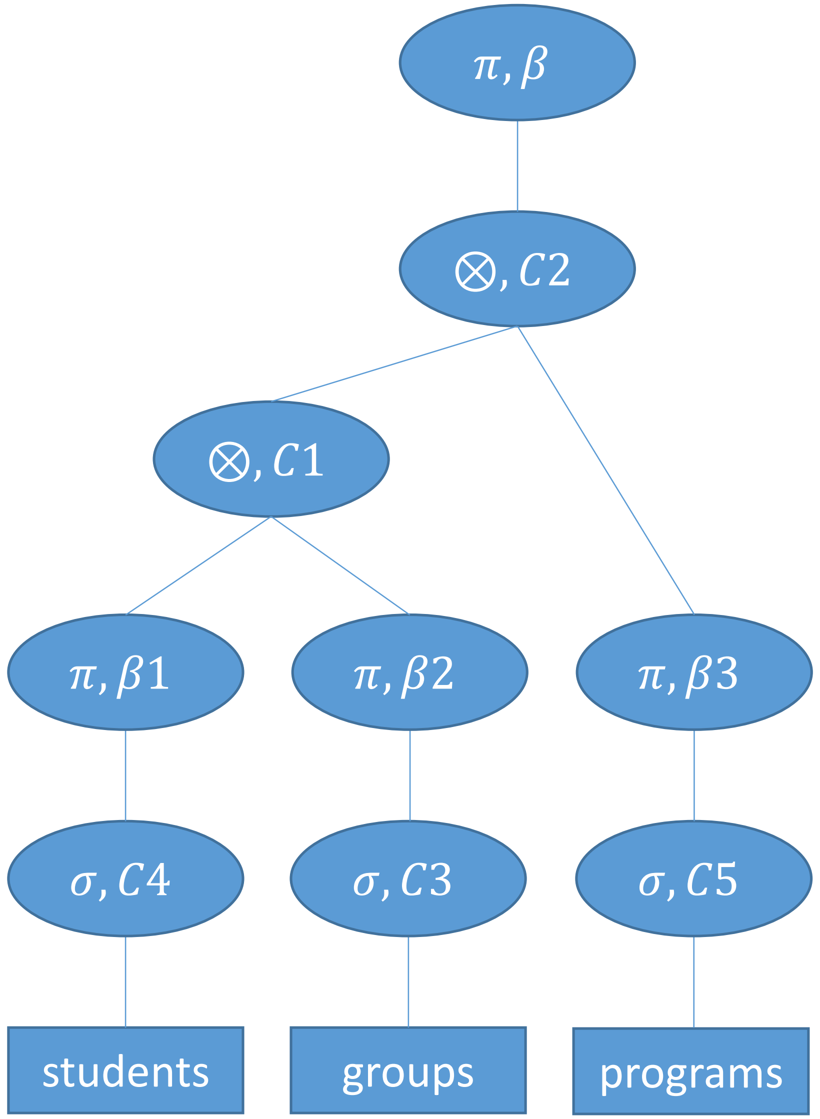
$$e = \pi_{\beta}(((\pi_{\beta 1}(\sigma_{C4}(students))) \otimes_{C1} (\pi_{\beta 2}(\sigma_{C3}(groups)))) \otimes_{C2} (\pi_{\beta 3}(\sigma_{C5}(programs))))$$

- 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

[Si10] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, McGraw-Hill, 2010, http://codex.cs.yale.edu/avi/db-book/

[Kn76] KNUTH, D.E., Tratat de programare a calculatoarelor. Sortare și căutare. Ed. Tehnică, București, 1976

[Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book, Prentice Hall Press, 2008