Algorithms for Executing and Visualizing the Basic Relational Operations for Purposes of a Virtual Laboratory

Elitsa ARSOVA, Silyan ARSOV, Angel SMRIKAROV
Dept. of Computer Systems and Technologies, University of Rousse
Rousse, 7017, Bulgaria

ABSTRACT

Studying a "Databases" course includes understanding of complex notions and methods. Students' commonly expressed difficulties in the subject suggest that traditional approaches are not appropriate for some learners. In this paper, we present developed algorithms for execution of some operations on relations: projection, selection, union, difference, Cartesian product, intersection and natural join. We also report and explain the evaluation of algorithms' complexity. Finally, we mention program modules, performing the operations above, which are implemented as part of a developing Virtual Laboratory (VL).

Key words: Virtual Laboratory; Relational Operations; Algorithms; Algorithms' Complexity; Databases.

1. INTRODUCTION

Some students have difficulties learning a "Databases" course especially when they have to understand theoretical notions and data models. Such a notion is the algebraic notation, called relational algebra, where queries are expressed by applying specialized operators to relations. The relationship between operations on relations and SQL queries is hard to understand, because the realization of the queries on logical level is an official secret. Algorithms for execution of the family of operations on relations are not available either. This is the reason for proposing algorithms based on the definitions of respective operations on relations.

The idea is to design and implement the algorithms in an educational computer-tool that will be a part of the VL on "Databases" course. The algorithms are developed specially for simulating the execution of relational operations on logical level for educational purposes. They are represented as block schemas and do not have an aim to show the implementation of a real database management system (DBMS).

During the past years Virtual Learning Environments (VLE) have rapidly influenced the educational process as they are considered to substitute traditional methods of learning. At present, there is a trend to interchange and combine full-time with distance form of study [11]. This tendency seems to be a substantial component of the educational policy of many leading universities. Some of the Bulgarian universities that *implemented* e-learning systems for *enhanced learning* are the University of Rousse – the eLSe system [12], the University of Sofia – the ARCADE system [9], the University of Plovdiv – PeU [6], the Technical University-Sofia[4], etc.. The researches in the field of VLs at the Department of "Computer Systems and Technologies" in the University of Rousse proceed with implementation of VL on the courses "Computer Organizations" [1] and "Analysis and Synthesis of Logical Schemes", named Digital Logic Design Virtual Laboratory (DLDVL) [13]. Interactive models of different Central Processing Unit (CPU) blocks are created within the framework of the VL on "Computer Organization". Tools for logical schemes design and test system are developed in DLDVL.

Learning a "Databases" course includes aspects from conceptual and theoretical knowledge to practical development and implementation skills [2]. The choice of topics for several courses and practical seminars on database systems is discussed in [7], where the idea of the environment is to achieve a balance between theory and practice. An interactive educational multimedia system based on the virtual apprenticeship model for the knowledge- and skills-oriented Web-based education of students on "Databases" course is presented in [2]. A main aspect of the system is combining knowledge, learning and skills training in an integrated environment and the authors show that tool-mediated independent learning and training in an authentic setting is an alternative to the traditional classroom-based approaches. According to [10] database representation is an important factor in database use and the interaction between a database structure representation and a query language may dramatically affect database learning and use.

2. THEORETICAL BASIS

Definition of relational algebra: Let U is a set of attributes, called universium, D- a set of domains and dom- an entire function of U in D. Let further $R = \{R_1, R_2, .., R_n\}$ is a set of different schemes of the entities, $d = \{r_1, r_2, .., r_n\}$ is a set of all relations, where r_i is a relation with scheme R_i, $1 \leq i \geq n$. Θ is a set of binary relations over the D domains, containing the arithmetic comparative operations on every domain.

Relational algebra over U, D, dom, R, d, Θ is called seven positioned tuple $\Re = (U, D, dom, R, d, \Theta, O)$, where O is a set of the operators for *projection*, *selection*, *union*, *difference*, *Cartesian product*, *intersection* and *join*, that uses relation Θ.
The algebra expression over $\Re$ is called every expression that is built correctly from relations related to d and from constant relations with schemes U using the operators from O [3].

3. ALGORITHMS FOR EXECUTION OF THE OPERATIONS ON RELATIONS

Projection. The idea behind this operation is that we take a relation R, remove some of the components (attributes) and/or rearrange some of the remaining components. If R is a relation

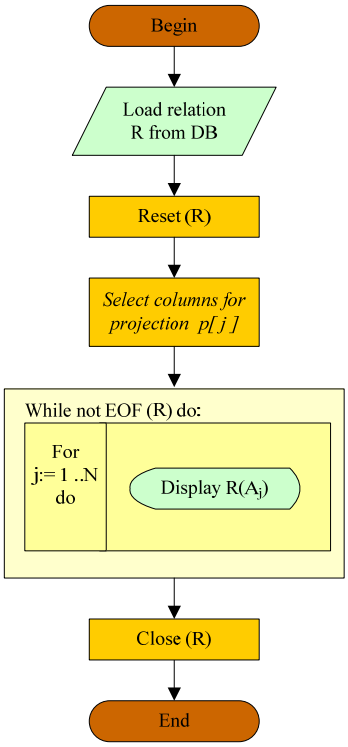

Fig. 1. An algorithm for execution of the relational operation *projection*

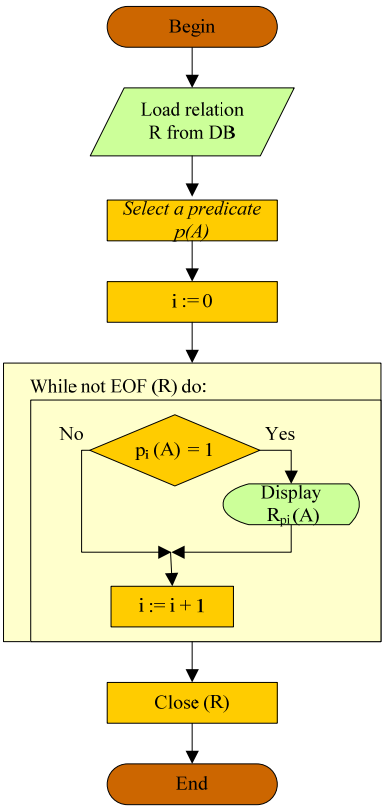

Fig. 2. An algorithm for execution of the relational operation *selection*

of arity k, we let $\pi_{i_1,i_2,..,i_m}(R)$, where i_j's are distinct integers in the range 1 to k, denote the *projection* of R onto components $i_1, i_2, .., i_m$, that is, the set of m – tuples $a_1, a_2, .., a_m$ such that there are some k – tuples $b_1, b_2, .., b_k$ in R for which $a_i = b_{i_j}$, for $j = 1,2, .., m$ [8]. The algorithm of operation *projection* is presented on fig. 1.

Selection. Let F be a formula involving [8]:

1) Operands that are constants or component numbers; component i is represented by $\$i$,
2) The arithmetic comparison operators $<, =, >, \leq, \neq, \geq$, and
3) The logical operators $\wedge (and), \vee (or), and, \neg (not)$.

Then $\sigma F(R)$ is a set of tuples μ in R such that when, for any i, we substitute the i^{th} component of μ for all occurrences of $\$i$ in formula F, the formula F becomes true. The algorithm of operation *selection* is presented on fig. 2.

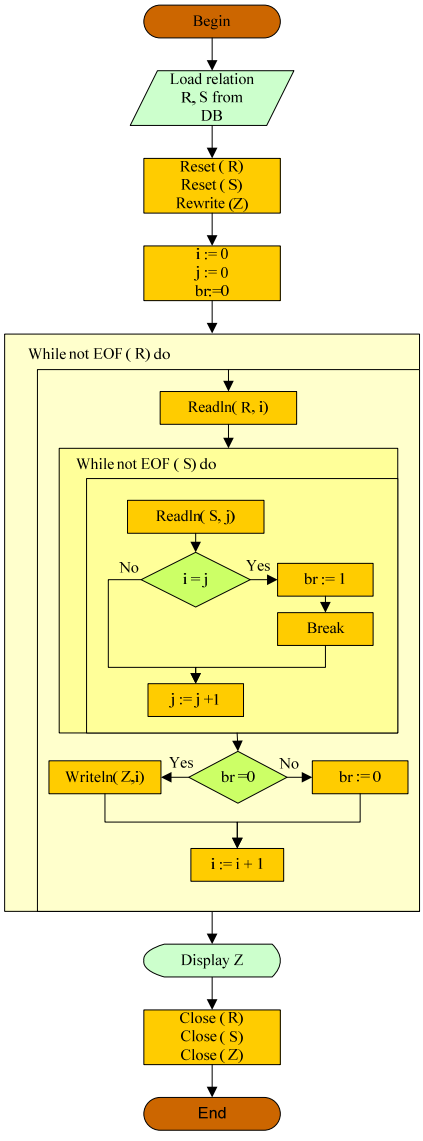

Fig. 3. An algorithm for execution of the relational operation *difference*

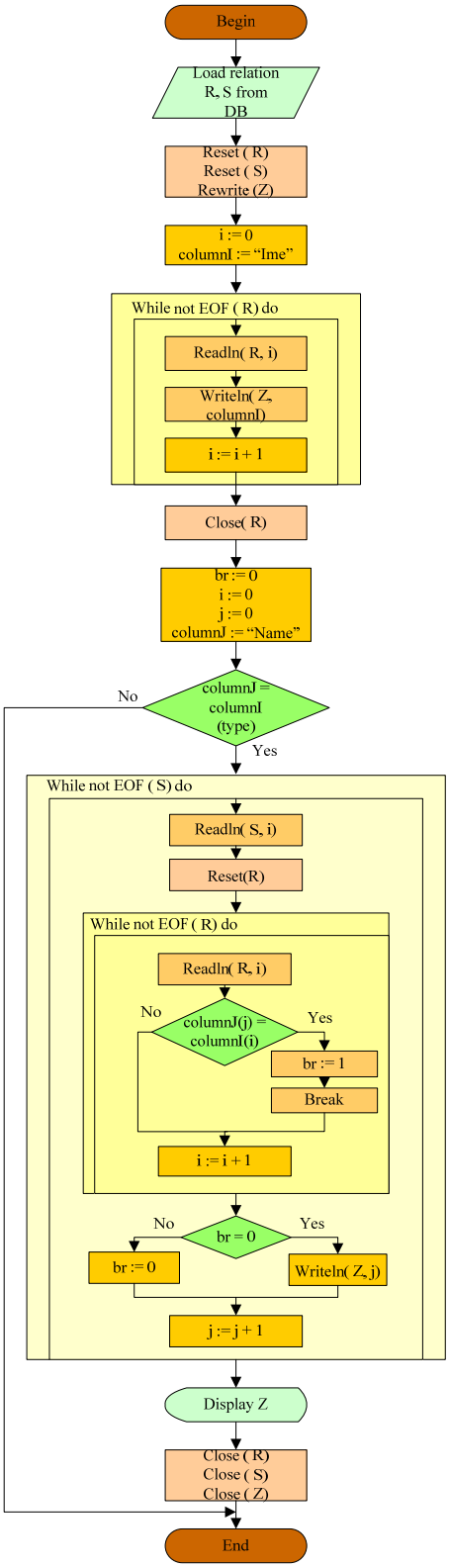

Fig. 4. An algorithm for execution of the relational operation *union*

Difference. The *difference* [5] of relations R *and* S, denoted $R - S$, is the set of tuples in R but not in S (fig. 3).

Union. The *union* [5] of relations R *and* S, denoted $R \cup S$, is the set of tuples that are in R *or* S or both (fig. 4).

Cartesian product. Let R *and* S be relations of arity k_1 *and* k_2, respectively. Then $R \times S$, the Cartesian product of R *and* S, is the set of all possible $(k_1 + k_2)$ – tuples whose first k_1 components from a tuple in R and whose last k_2 components from a tuple in S [8]. The algorithm of operation *Cartesian product* is presented on fig. 5.

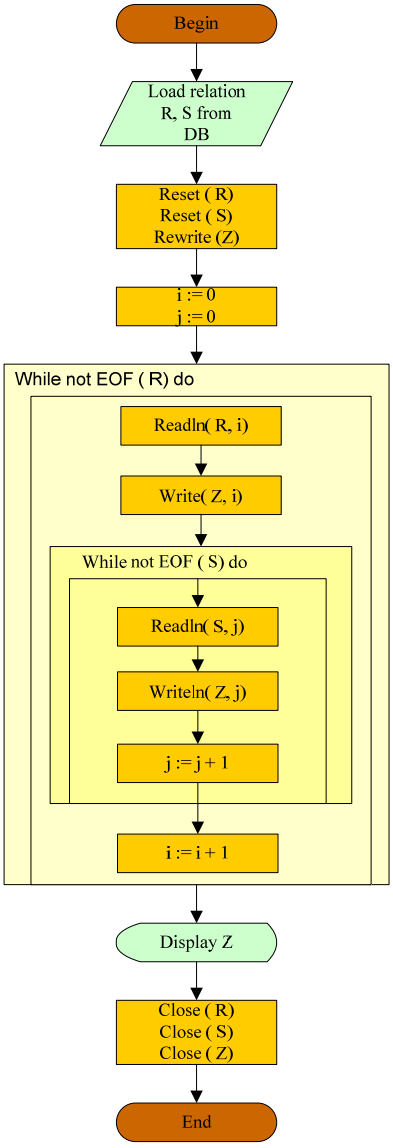

Fig. 5. An algorithm for execution of the relational operation *Cartesian product*

Intersection. The *intersection* of relations R *and* S, denoted $R \cap S$, is the set of elements that are in both relations [5]. Since *intersection* can be written in terms of set-difference, it is not a fundamental operation. The algorithm is presented on fig. 6.

For the execution of operations *union*, *difference and intersection* the relations R *and* S must have schemas with identical sets of attributes, and types (domains) for each attribute must be the same in R *and* S.

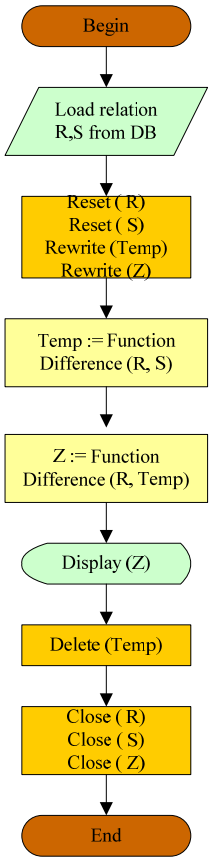

Fig. 6. An algorithm for execution of the relational operation *intersection*

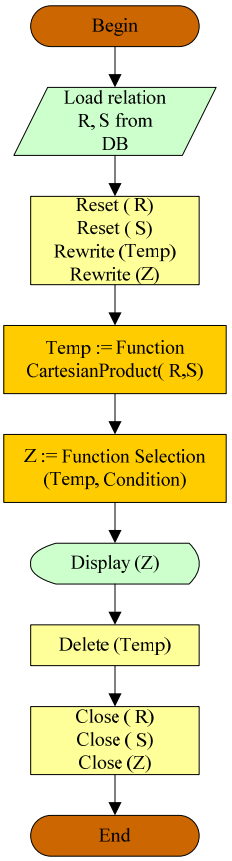

Fig. 7. An algorithm for execution of the relational operation *natural join*

Natural join. The natural join, written $R \bowtie S$, is applicable only when both R *and* S have columns that are named by attributes. To compute $R \bowtie S$ we:

1. Compute $R \times S$;
2. For each attribute A that names both a column in R and a column in S select from $R \times S$ those tuples whose values agree in the columns for $R.A$ and $S.A$, where $R.A$ is the name of the column of $R \times S$ corresponding to the column A of R, and $S.A$ is defined analogously;
3. For each attribute A above, project out the column $S.A$ and call the remaining column, $R.A$, simply A.

Formally then, if $A_1, A_2, .., A_k$ are all the attribute names used for both R *and* S, we have $R \bowtie S = \pi_{i_1, i_2, .., i_m}(\sigma_{R.A_1=S.A_1 \wedge ... \wedge R.A_k=S.A_k}(R \times S))$, where $i_1, i_2, \ldots, i_m$ is the list of all components of $R \times S$, in order, except the components $S.A_1, S.A_2, \ldots, S.A_k$ [8]. The algorithm of operation *natural join* is presented on fig. 7.

4. IMPLEMENTATION OF THE ALGORITHMS

To show that the developed algorithms in fact work at a physical level, we implement them in a web-based programming system. We disregard a full storage manager with a buffer scheme to facilitate the students' understanding on relational theory. The system is experimental and its aim is to visualize the execution of basic operations on relations at a logical level (i.e. how rows and columns are excluded from the output).

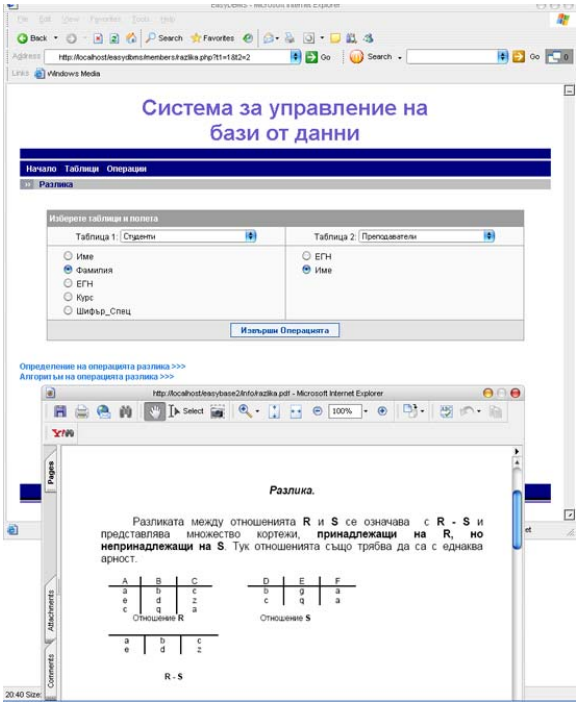

Fig. 8. A user interface for execution of the relational operation *difference* and its definition

The idea is to demonstrate the execution of the respective relational operations. The demonstration presents the result from the execution of chosen operations with indicated operands.

A user interface for execution of the relational operation *difference* is presented on fig. 8. The given page enables users to select tables (relations), among which the operation will be performed. After selecting a table, all of its attributes are displayed and some of them must be chosen. Depending on the operation, we choose one or several attributes.

Furthermore, the system has a possibility to display learning materials, as definitions and algorithms, in additional windows. The link "Definition of the operation *difference*" opens a PDF file in a new browser window (fig. 8). By this means, the user could examine the theoretical part and choose simultaneously the attributes that could participate in a given query.

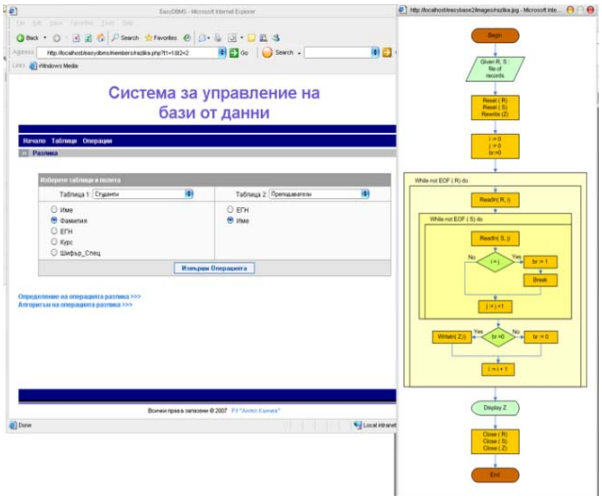

Fig. 9. A user interface for execution of
the relational operation *difference* and its algorithm

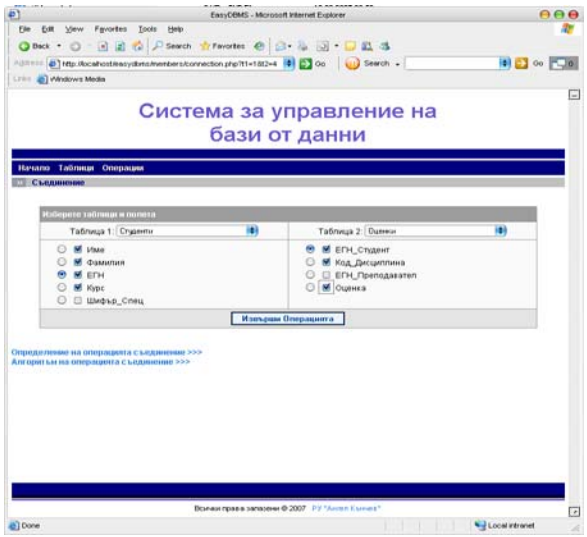

Fig. 10. A user interface for execution of
the relational operation *natural join*

Likewise, the link "Algorithm of the operation *difference*" opens in a new browser window and visualizes the sequence of steps for execution of the operation (fig. 9).

The user interface for execution of the operation *natural join* is distinguished from others (fig. 10). Its form consists of an additional field which indicates the attributes that are common for the two tables i.e. specifying primary and foreign keys of the relations.

5. ESTIMATION OF ALGORITHMS' COMPLEXITY

Projection. Let have a sequence of attribute values $A: a_1, a_2, \dots, a_n$.
The *projection* can be executed by reading N times consecutively the values of the chosen attributes A for the projection, where N is the number of values of A, and transferring to the new array that consists of the result from operation *projection*.
The Eq. (1) presents the *projection* algorithm complexity:

$$T_{executions} = O(N), \quad (1)$$

where $T_{execution}$ is the number of executions of the "*read/write*" operations.

Selection. Let have a sequence of attribute values $A: a_1, a_2, \dots, a_n$.

Algorithm for binary search in ordered sequence of data values.
This algorithm is used in cases when the sequence of attributes values $a_1, a_2, \dots, a_n$ is ordered. We should find whether an element x is in this sequence. The idea of the binary search is the following. An element x is compared with an element in the middle of the sequence - a_{middle}. The search is continuing in the same way on the left half of array if $x < a_{middle}$. The search is continuing on the right half if $x > a_{middle}$ and the search will stop when $x = a_{middle}$.

The estimation of the algorithm complexity (Eq. (2)) in regard to operation *selection* is the following:

$$T_{\min\ \_comparisons} = 1;\ T_{\max\ \_comparisons} = \log_2 N, \quad (2)$$

where $T_{comparisons}$ is the number of comparisons.

Algorithm for sequential search in not ordered sequence of data values.
The estimation of the sequential search algorithm complexity (Eq. (3)) regarding to operation *selection* is the following:

$$T_{\min\ \_comparisons} = 1;\ T_{\max\ \_comparisons} = N, \quad (3)$$

where $T_{comparisons}$ is the number of comparisons.

Union. Let have a sequence of attributes values: $A: a_1, a_2, \dots, a_n$ and $B: b_1, b_2, \dots, b_m$.
The *union* can be executed by reading N times consecutively the values of attribute A and transfer them into a new array that consists of the result from operation *union.* Next, we read M times attribute B values, compare values of B with values of A to avoid repetitions and again transfer them into the already existing array consisting of the result from operation *union*.

The next Eq. (4) presents the estimation of *union* algorithm complexity:

$$T_{executions} = O(N + N * M) = O(N(M + 1)), \quad (4)$$

where $T_{execution}$ is the number of executions of the "*read/write*" and comparison operations.

Difference. Let have a sequence of attributes values: $A: a_1, a_2, \dots, a_n$ and $B: b_1, b_2, \dots, b_m$.
The *difference* operation can be executed by reading N times consecutively the attribute A values and all of its values are compared M times with the values of attribute B. If a given value of A is not equal to a value of B, the value of A is transferred into a new array consisting of the result of operation *difference*.

The estimation of the algorithm complexity (Eq. (5)) for execution of operation *difference* is the following:

$$T_{executions} = O(N * M), \quad (5)$$

where $T_{execution}$ is the number of executions of the "*read/write*" and comparison operations.

Cartesian product. Let have a sequence of attributes values: $A: a_1, a_2, \dots, a_n$ and $B: b_1, b_2, \dots, b_m$.
The *Cartesian product* can be executed by reading N times consecutively the attribute A values and by concatenating all of its values with all values of attribute B. The second operation is executed M times. The outcome is a new array containing the result of operation *Cartesian product*.

The next Eq. (6) presents the estimation of *Cartesian product* algorithm complexity:

$$T_{executions} = O(N * M), \quad (6)$$

where $T_{execution}$ is the number of execution of the "*read/write*" operations.

Intersection. Let have a sequence of attributes values: $A: a_1, a_2, \dots, a_n$ and $B: b_1, b_2, \dots, b_m$.
The *intersection* operation can be executed by reading N times consecutively the attribute A values and all of its values are compared M times with values of attribute B. If a given value of A is equal to a value of B, the value of A is transferred into a new array consisting of the result of operation *intersection.*

The estimation of the algorithm complexity for execution of operation *intersection* is the following (Eq. (7)):

$$T_{executions} = O(N * M), \quad (7)$$

where $T_{execution}$ is the number of execution of the "*read/write*" and comparison operations.

Natural join. Let have a sequence of attributes values: $A: a_1, a_2, \dots, a_n$ and $B: b_1, b_2, \dots, b_m$.
The *natural join* operation can be executed as a sequence of two operations: Cartesian product and selection. The estimation of the *Cartesian product* algorithm complexity is

$$T_{executions(CP)} = O(N * M).$$

The selection operation is executed after transferring the result from the Cartesian product operation execution to a new array consisting of $N * M$ values. The estimation of the sequential search algorithm complexity in regard to operation *selection* is the following:

$$T_{executions(S)} = O(N * M),$$

i.e. the number of array rows that are the result from Cartesian product operation.

The sum of the estimations of Cartesian product and selection algorithms complexity forms the estimation of natural join algorithm complexity (Eq. (8)).

$$T_{execution(NJ)} = T_{execution(CP)} + T_{execution(S)}$$
$$T_{execution(NJ)} = O(N * M + N * M) = 2 * O(N * M) \quad (8)$$

6. CONCLUSIONS AND FUTURE WORK

Algorithms based on the definitions of the operations *projection, selection, union, difference, Cartesian product, intersection and natural join* are developed with a purpose to explain and visualize the operations on relations explicitly. Learner knowledge is elaborated by presenting complex definitions as algorithms or a sequence of simple operations. The algorithms are used for the implementation of program modules for the purposes of a VL on "*Databases"* course. Through the implementation of the program modules based on algorithms, developed by the authors, has proved their correctness. Results from the testing of the algorithms in the VL with a typical group of students will be published in a separate paper.

7. REFERENCES

[1] A. Vasileva, A. Smrikarov, T. Hristov, "A Conceptual Model of a Virtual Laboratory on "Computer Organization"", **Proceedings of CompSysTech'02**, Sofia, Bulgaria, 20-21 June, 2002.

[2] C. Pahl , R. Barrett , Claire Kenny, "Supporting Active Database Learning and Training through Interactive Multimedia", **Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education**, Leeds, United Kingdom, June 28-30, 2004.

[3] D. Maier, **The Theory of Relational Databases,** Rockville, Md.: Computer Science Press, 1983.

[4] E. Shoikova, V. Denishev, I. Pandiev, "Development of an eLearning Architecture Based on Microsoft Class Server", **International Conference "New Technologies in Higher Education and Training"**, Sofia, 6-17 May, 2003.

[5] G. Molina, J. Ullman, J. Widom, **Database Systems: The Complete Book,** Prentice Hall, New Jersey, USA, 2002.

[6] G. Totkov, R. Doneva, "Computerized environment for integrated maintenance of distance education course modules", **Proceedings of the 1998 EDEN Conference**, Vol. 2, Italy, 1998.

[7] J. Peneva, G. Tuparov, "An Approach of Teaching Data Management", **Proceedings of CompSysTech'05**, Varna, Bulgaria, 16-17 June, 2005.

[8] J. Ullman, **Principles of database and knowledge-base systems**, Vol. 1 Classical database systems, Computer Science Press, 1988.

[9] K. Stefanov, S. Stoyanov, R. Nikolov, "Design Issues of a Distance Learning Course on Business on the Internet", **JCAL (Journal of Computer Assisted Learning)**, Vol. 14, No. 2, 1998.

[10] R. Leitheiser, S. March, "The Influence of Database Structure Representation on Database System Learning and Use", **Journal of Management Information Systems**, Vol. 12, Issue 4, March 1996, p.187 – 213.

[11] S. Ivanov, J. Peneva, "Distance Learning Courses in Computer Science – Initiation and Design", **Proceedings of CompSysTech'07**, Rousse, Bulgaria, 14-15 June, 2007.

[12] T. Hristov, S. Smrikarova, A. Vasileva, A. Smrikarov, "An Approach to Development of an e-Learning Software Platform", **Proceedings of CompSysTech'02**, Sofia, Bulgaria, 2002.

[13] V. Mateev, S. Todorova, A. Smrikarov, "Test Construction and Distribution in Digital Logic Design Virtual Laboratory", **Proceedings of e-Learning Conference'07 Computer Science Education**, Istanbul, Turkey, 27-28 August, 2007.