User-Defined Financial Functions for MS SQL Server

Jolana Gubalova, Petra Medvedova
Department of Quantitative Methods and Information systems
Faculty of Economics, Matej Bel University
Banska Bystrica, Slovakia

***Abstract*—The paper deals with mathematical preparation and subsequent programming of various types of financial functions with using of Transact-SQL in Database Management System MS SQL Server. Financial functions are used to automate calculations in the area of Financial Economics. In MS SQL Server, any financial functions are not offered for financial data processing, how such as in program MS Excel. We emphasize that we have used a different calculation methods to create financial formulas, not those used in Excel. If users want to work with some special functions, there is a possibility to prepare User-Defined Functions (UDFs). The use of UDFs will make it easier to work on financial calculations in large databases.**

Keywords*—*Financial economics; user-defined functions; financial functions; database management system; structure query language; transact-SQL

I. Introduction

Aggregate queries over big economic relational databases, prepared with using of Structure Query Language (SQL) and special programs belong to the most used tools in the area of Business Intelligence (BI) and Data Analytics (DA). One of the most important Relational Database Management Systems (RDBMS) for data saving, processing and analyzing, in the area of huge corporate or financial databases, is MS SQL Server. SQL Server runs on Transact – SQL (T-SQL), a set of programming extensions, that add several features to standard SQL, including transaction control, error handling, row processing and declared variables. SQL Server also allows stored procedures to be defined. Functions are a special type of stored procedures. They accept parameters, perform some sort of action and return a result. Functions do all of this with no side effects [14]. As Simhadri, V., at all said [12], queries containing user-defined functions (UDFs) are widely used, since they allow queries to be written using a mix of imperative language constructs and SQL, thereby increasing the expressive power of SQL; further, they encourage modularity, and make queries easier to understand. Writing user-defined functions or stored procedures presents common way in application development using a relational database management system. It allows to embed application code inside of RDBMS [15]. SQL Server provides numerous types of built-in scalar functions, for example, there are many built-in mathematical functions, date functions, string functions or aggregate functions. The types of user-defined functions (UDFs), that SQL Server supports, are scalar (return a single value) and table-valued (return a table). Most commercial SQL database systems support user-defined functions that can be used in WHERE clause filters, SELECT list items, or in sorting/grouping clauses. Often, user-defined functions are used as inexact search filters and then the filtered rows are sorted by a relevance measure [8]. Running analytics computation inside a database engine through the use of UDFs (User Defined Functions) has been investigated, but not yet become a scalable approach due to several technical limitations. One limitation lies in the lack of generality for UDFs to express complex applications and to compose them with relational operators in SQL queries. Another limitation lies in the lack of systematic support for a UDF to cache relations initially for efficient computation in multi-calls. Further, having UDF execution interacted efficiently with query processing requires detailed system programming, which is often beyond the expertise of most application developers [4]. Ordonez, C., at all described in [10], [11] vector and matrix operations programmed with UDFs in a relational DBMS and a data mining system based on SQL queries and UDFs for relational databases. Sousa, M., at all [13] dealt with consolidation of queries with UDFs.

UDFs can also be used in Excel. Lester in [6] recommended them as alternative methods to perform duct calculations.

II. Objective and Methods

In SQL Server, any financial functions are not offered for financial data processing, how such as in program MS Excel. This fact we felt like a big shortage in processing of financial data. Because of this problem, we decided to prepare main financial functions, available in program MS Excel, also in MS SQL Server with using of program extension T-SQL. There were particularly financial functions: for calculation of the future value of an investment based on a constant interest rate, for returning the number of periods for an investment based on periodic, constant payments and a constant interest rate, for calculation of the payment for a loan based on constant payments and a constant interest rate, or for calculation of the present value of a loan or an investment, based on a constant interest rate. The concepts of financial mathematics are described in sources [1] – [3], [9].

Finally we compared the speed and efficiency of work with classical formulas and UDFs in SQL Server 2012 with using of special tools - Execution plan and Client Statistics. Execution plans display how the database engine navigates tables and uses indexes to access or process the data for a query or other DML (Data Manipulation Language) statement, such as an update [7]. This graphical approach is very useful for understanding the performance characteristics of a query.

Client Statistics is SQL Server data tool which is very helpful in determining the statistics that how much data received from server to the client side. It means, client statistics helps in analyzing the traffics load like packets/bytes sent and received at client – server side. When we run a script or query in T-SQL editor, we can enable Client statistics to collect statistics like application profile, time statistics and network statistics which help in checking the efficiency of the script.

III. Results and Discussion

In the following, we approached the basic knowledge and relationships of interest and rent numbers that we later used. We only dealt with compound interest when the interest is added to the original capital and the sum is further capitalized. In all considerations, we considered overdue (decursive) capitalization, in other words - interest paid at the end of the interest period. Throughout the text, we used the following symbols in Table I.

TABLE I. List of arguments

Financial mathematics	**T-SQL**	**Meaning**
FV	@FV	Future Value of a capital
PV	@PV	Present Value of a capital
PMT	@PMT	Payment based on regular constant payments
i	@RATE	Interest Rate per year
m	@NPER	Number of conversions per year
n	@YEARS	Number of years
p	@NPAY	Number of payments per year
T	@TYPE	Value representing the timing of payment T=1 payment at the beginning of the period T=0 payment at the end of the period

We calculated the future value of the initial capital at compound interest over n years based on the following formula by [5].

$$FV = PV \cdot (1+i)^n. \tag{1}$$

In practice, it is common that the interest rates are considered more often than once a year, and then we talk about compound interest capitalization with conversions. We call the period between the two following interest charges conversion. Interest is generally charged m -times annually. The future value of capital in compound interest with conversions in n years was determined based on the following formula by [5]

$$FV = PV \cdot \left(1+\frac{i}{m}\right)^{m \cdot n}. \tag{2}$$

In real-life economics, we often encounter a system of regularly repeated payments. This sequence of regularly repeated payments is called a rent or cash flow. In this analysis, we dealt only with constant, unconditional, temporary, immediate-term rents (cash flows). With constant rent, the amount of the individual payments does not change (remains unchanged). Unconditional or sure rent is a rent, where individual rent payments are not subject to any conditions. Temporary or terminal rent has the finite number of payment. We are talking about immediate rent, if the first payment is made at the beginning or end of the first rent period. When considering a p -term rent, p determines the number of payments per year. If the payments are always made at the end of the time period, we are talking about a strenuous (overdue, post-term) rent, if the payments are always made at the beginning of the period, we are talking about the pre-term rent. The future value of an annuity (rent payments, cash flow) is the sum of the future values of all annuity payments calculated at the end of the n -th year.

For the future value of a p -term strenuous (overdue, post-term) rent after years, the following formula applies by [5]

$$FV = PMT \cdot \frac{\left(1+\frac{i}{m}\right)^{m \cdot n} - 1}{\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1}. \tag{3}$$

The future value of a p -term rent, provided a pre-term (pre-paid) payment after n years was determined based on the following formula by [5]

$$FV = PMT \cdot \left(1+\frac{i}{m}\right)^{\frac{m}{p}} \cdot \frac{\left(1+\frac{i}{m}\right)^{m \cdot n} - 1}{\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1}. \tag{4}$$

We were interested in the future value of the initial capital provided, that we regularly deposited payments to the initial capital p -times per year for n years with annual interest rate i and m conversions per year.

A. Future Value of a Series of Payments

At first we considered the overdue (post-term) rent, so we paid the payments at the end of p -th of the year each year. Next we considered the pre-term rent, so we paid the instalments at the beginning of p -th of the year each year. The future value in our case we determined as the sum of the future value of compound interest with conversions and the future value of the p -term rent by using formulas (2) and (3) for post-term rent and by using formulas (2) and (4) for pre-term rent, which resulted in the following formulas in Table II.

TABLE II. FUTURE VALUE – FORMULAS AND PROGRAM CODE

T=0

$$FV = PV \cdot \left(1+\frac{i}{m}\right)^{m \cdot n} + PMT \cdot \frac{\left(1+\frac{i}{m}\right)^{m \cdot n} - 1}{\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1}$$

T=1

$$FV = PV \cdot \left(1+\frac{i}{m}\right)^{m \cdot n} + PMT \cdot \left(1+\frac{i}{m}\right)^{\frac{m}{p}} \cdot \frac{\left(1+\frac{i}{m}\right)^{m \cdot n} - 1}{\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1}$$

```
--FV
CREATE FUNCTION  FV (@RATE FLOAT, @YEARS FLOAT, @PMT FLOAT,@PV FLOAT, @NPER FLOAT, @NPAY FLOAT, @TYPE INT)
RETURNS MONEY
AS
BEGIN
DECLARE @FV MONEY
IF @TYPE = 0
SET @FV = @PMT*((POWER(1+@RATE/@NPER,@YEARS*@NPER)-1)/(POWER(1+@RATE/@NPER,@NPER/@NPAY)-1))+
@PV*POWER(1+@RATE/@NPER,@NPER*@YEARS)
IF @TYPE = 1
SET @FV = @PMT*((POWER(1+@RATE/@NPER,@YEARS*@NPER)-1)/(POWER(1+@RATE/@NPER,@NPER/@NPAY)-
1))*POWER(1+@RATE/@NPER,@NPER/@NPAY)+ @PV*POWER(1+@RATE/@NPER,@NPER*@YEARS)
RETURN @FV
END
```

B. Present Value of a Capital

From the formulas for the future value FV (2), (3), (4), we expressed the initial deposit which is needed in the periodic payments to accumulate the future value using the following expressions in Table III.

TABLE III. PRESENT VALUE OF A CAPITAL – FORMULAS AND PROGRAM CODE

T=0

$$PV = FV \cdot \left(1+\frac{i}{m}\right)^{-m \cdot n} - PMT \cdot \frac{1-\left(1+\frac{i}{m}\right)^{-m \cdot n}}{\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1}$$

T=1

$$PV = FV \cdot \left(1+\frac{i}{m}\right)^{-m \cdot n} - PMT \cdot \left(1+\frac{i}{m}\right)^{\frac{m}{p}} \cdot \frac{1-\left(1+\frac{i}{m}\right)^{-m \cdot n}}{\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1}$$

```
CREATE FUNCTION PV (@RATE FLOAT, @YEARS FLOAT, @PMT FLOAT, @FV FLOAT, @NPER FLOAT, @NPAY FLOAT, @TYPE INT)
RETURNS MONEY
AS
BEGIN
DECLARE @PV MONEY
IF @TYPE = 0
SET @PV =(@FV*POWER(1+@RATE/@NPER,-@NPER*@YEARS) - @PMT*(1-POWER(1+@RATE/@NPER,-
@NPER*@YEARS))/(POWER(1+@RATE/@NPER,@NPER/@NPAY)-1))
IF  @TYPE = 1
SET @PV = @FV*POWER(1+@RATE/@NPER,-@NPER*@YEARS) - @PMT*POWER(1+@RATE/@NPER,@NPER/@NPAY)*(1-
POWER(1+@RATE/@NPER,-@NPER*@YEARS))/(POWER(1+@RATE/@NPER,@NPER/@NPAY)-1)
RETURN @PV
END
```

C. Payment based on Regular Constant Payments

We determined the amount of a regular instalment PMT which would give us with the initial deposit PV after n years a future value FV. It was enough to express from formula for future value FV (2), (3), (4), p-term payment after n years to express the unknown PMT and we have got the following formulas in Table IV.

TABLE IV. PAYMENT BASED ON REGULAR CONSTANT PAYMENTS – FORMULAS AND PROGRAM CODE

T=0

$$PMT = \left[FV - PV \cdot \left(1+\frac{i}{m}\right)^{m \cdot n} \right] \cdot \frac{\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1}{\left(1+\frac{i}{m}\right)^{m \cdot n} - 1}$$

T=1

$$PMT = \left[FV - PV \cdot \left(1+\frac{i}{m}\right)^{m \cdot n} \right] \cdot \frac{\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1}{\left(1+\frac{i}{m}\right)^{m \cdot n} - 1} \cdot \left(1+\frac{i}{m}\right)^{-\frac{m}{p}}$$

```
--PMT
CREATE FUNCTION PMT (@RATE FLOAT,@YEARS FLOAT, @PV FLOAT,@FV FLOAT, @NPER FLOAT, @NPAY FLOAT, @TYPE BINARY)
RETURNS MONEY
AS
BEGIN
DECLARE @PMT MONEY
IF @TYPE = 0
SET @PMT = (@FV-@PV*POWER(1+@RATE/@NPER,@NPER*@YEARS))*(POWER(1+@RATE/@NPER,@NPER/@NPAY)-
1)/(POWER(1+@RATE/@NPER,@NPER*@YEARS)-1)
IF @TYPE = 1
SET @PMT = (@FV-@PV*POWER(1+@RATE/@NPER,@NPER*@YEARS))*POWER(1+@RATE/@NPER,-
@NPER/@NPAY)*(POWER(1+@RATE/@NPER,@NPER/@NPAY)-1)/(POWER(1+@RATE/@NPER,@NPER*@YEARS)-1)
RETURN @PMT
END
```

TABLE V. NUMBER OF YEARS – FORMULAS AND PROGRAM CODE

T=0

$$n = \frac{1}{\ln\left(1+\frac{i}{m}\right)^{m}} \cdot \ln\left\{ \frac{FV \cdot \left[\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1 \right] + PMT}{PV \cdot \left[\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1 \right] + PMT} \right\}$$

T=1

$$n = \frac{1}{\ln\left(1+\frac{i}{m}\right)^{m}} \cdot \ln\left\{ \frac{FV \cdot \left[\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1 \right] + PMT \cdot \left(1+\frac{i}{m}\right)^{\frac{m}{p}}}{PV \cdot \left[\left(1+\frac{i}{m}\right)^{\frac{m}{p}} - 1 \right] + PMT \cdot \left(1+\frac{i}{m}\right)^{\frac{m}{p}}} \right\}$$

```
--YEARS
CREATE FUNCTION YEARS (@RATE FLOAT,@NPay FLOAT,@PV FLOAT,@FV FLOAT, @PMT FLOAT, @NPER FLOAT, @TYPE INT)
RETURNS FLOAT
AS
BEGIN
DECLARE @YEARS FLOAT
IF @TYPE = 0
SET @YEARS = LOG((((@FV*(POWER(1+@RATE/@NPER,@NPER/@NPay)-1)+@PMT))/(((@PV*(POWER(1+@RATE/@NPER,@NPER/@NPay)-
1)+@PMT)))
/LOG(POWER(1+@RATE/@NPER,@NPER))
IF @TYPE = 1
SET @YEARS = LOG((((@FV*(POWER(1+@RATE/@NPER,@NPER/@NPay)-
1)+@PMT*POWER(1+@RATE/@NPER,@NPER/@NPAY)))/(((@PV*(POWER(1+@RATE/@NPER,@NPER/@NPay)-
1)+@PMT*POWER(1+@RATE/@NPER,@NPER/@NPAY))))
/LOG(POWER(1+@RATE/@NPER,@NPER))
RETURN @YEARS
END
```

D. Number of Years

The same, on the basis of previous formulas (2), (3), (4), we have expressed a number of years n which are needed on obtaining of the future value FV . The corresponding formulas are as follows in Table V.

E. Number of Payments per Year

By expressing the number of payments per year p from future value formulas (2), (3), (4), we obtained the following formulas in Table VI.

F. The Comparison of Classical Calculations and Calculations with using UDFs

For comparison the speed and efficiency of work with classical formulas and UDFs we decided to use a training financial database with 1 048 575 records in DBMS MS SQL Server 2012. Firstly we run query with using of classical formula for calculation of Future Value and next with UDF Future Value. We also controlled Execution plans and Clients Statistics for this queries.

The Execution Plan consists of different operations and each operation has one output which is called the result set. The operations can have one or more inputs. There are many potential ways to execute a query thus SQL Server has to choose the most beneficial one. Client statistics helps in analyzing the traffics load like packets/bytes sent and received at client – server side. When we run a script or query in T-SQL editor, we can enable Client statistics to collect statistics like application profile, time statistics and network statistics which help in checking the efficiency of the query.

TABLE VI. NUMBER OF PAYMENTS PER YEAR– FORMULAS AND PROGRAM CODE

T=0	T=1
$p = \frac{\ln\left(1+\frac{i}{m}\right)^{m}}{\ln\left\{\frac{PMT\cdot\left[\left(1+\frac{i}{m}\right)^{m\cdot n}-1\right]}{FV-PV\cdot\left(1+\frac{i}{m}\right)^{m\cdot n}}+1\right\}}$	$p = \frac{\ln\left(1+\frac{i}{m}\right)^{m}}{-\ln\left\{1-\frac{PMT\cdot\left[\left(1+\frac{i}{m}\right)^{m\cdot n}-1\right]}{FV-PV\cdot\left(1+\frac{i}{m}\right)^{m\cdot n}}\right\}}$

```
-- Number of payments per year
CREATE FUNCTION NPay (@RATE FLOAT,@YEARS FLOAT,@PV FLOAT,@FV FLOAT, @PMT FLOAT, @NPER FLOAT, @TYPE INT)
RETURNS FLOAT
AS
BEGIN
DECLARE @NPay FLOAT
IF @TYPE = 0
SET @NPay = LOG(POWER(1+@RATE/@NPER,@NPER))/LOG((@PMT*(POWER(1+@RATE/@NPER,@YEARS*@NPER)-1))/(@FV-
@PV*POWER(1+@RATE/@NPER,@YEARS*@NPER))+1)
IF @TYPE = 1
SET  @NPay = LOG(POWER(1+@RATE/@NPER,@NPER))/-LOG(1-(@PMT*(POWER(1+@RATE/@NPER,@YEARS*@NPER)-1))/(@FV-
@PV*POWER(1+@RATE/@NPER,@YEARS*@NPER)))
RETURN @NPay
END
```

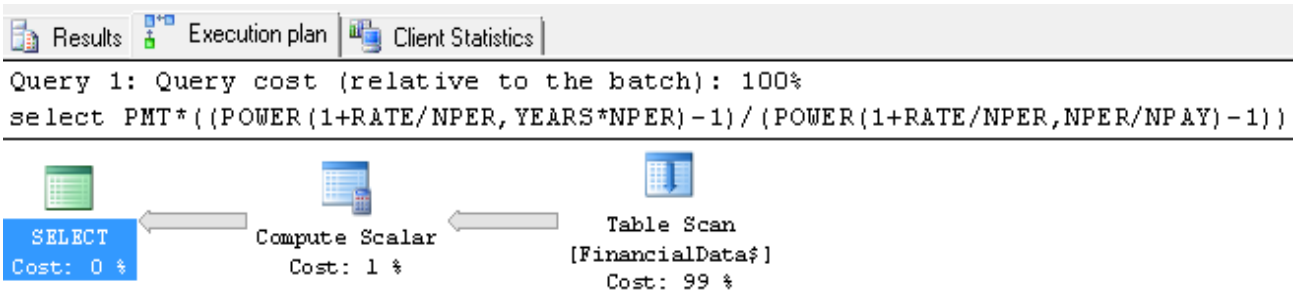

Fig. 1. Execution Plan of Classical Formula Future Value.

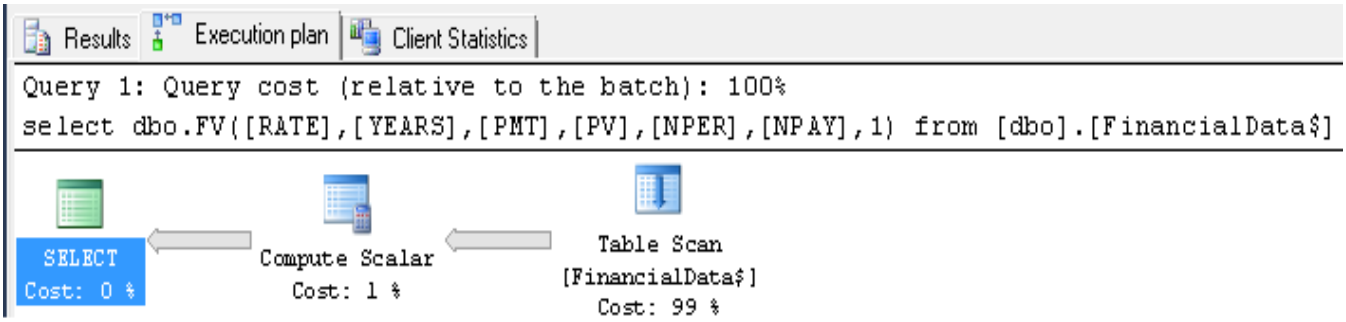

Fig. 2. Execution Plan of UDF Future Value.

As we can see at Figure 1, in case of classical formula Future Value, totally amount Query cost was spent by Table Scan of FinacialData (99%) and the rest (1%) was belong to Compute Scalar. The same situation was in case of UDF Future Value (Figure 2).

TABLE VII. COMPARISON OF CLIENT STATISTIC FOR CLASSICAL FORMULA AND UDF

	Classical Formula	**UDF**
Client Execution Time	16:43:38	16:52:49
Query Profile Statistics		
Number of SELECT statements	1	1
Rows returned by SELECT statements	1048575	1048575
Network Statistics		
Number of server roundtrips	3	1
TDS packets sent from client	3	1
TDS packets received from server	2572	2569
Bytes sent from client	504	210
Bytes received from server	1,052478E+07	1,050631E+07
Time Statistics [ms]		
Client processing time	1178,6	2797,8
Total execution time	1185,4	2806,8
Wait time on server replies	6,8	9,0

If we compared Client Statistics of classical formula Future Value and Client Statistics of UDF Future Value (Table VII.), we can stated, that *Query* Profile *Statistics* is the same in both cases.

But the difference is in Network Statistics – item *Bytes sent from client* is bigger at Classical Formula Future Value, because in case of UDF, client sent only data and formula were prepared on server side.

But there is significant difference in Client processing time. This attribute is almost 2,37 times higher for the UDF than for the Classical formula.

IV. CONCLUSIONS

The comparison classical calculations and calculations with using UDFs showed, that UDFs don't bring acceleration of the computation process opposite to the classical formulas. On the contrary, with using of the UDF the calculation process takes longer. However, working with them is simpler and more comfortable than with classical formulas, because they encourage modularity and make queries easier to understand. This is main reason they are popular among users in financial area. Stored procedures and UDFs can be prepared also in other extended DBMS, such as Oracle (Pragma UDFs), mySQL, PostgreSQL, DB2, Informix, etc.

REFERENCES

[1] Baz, J., Chacko, G., (2004). Financial derivatives: Pricing, applications, and mathematics. Publisher: Cambridge University Press, ISBN: 978-051180664-3;978-052181510-9.

[2] Dworsky, L.N., (2009). Understanding the Mathematics of Personal Finance: An Introduction to Financial Literacy. Publisher: John Wiley and Sons, ISBN: 978-047049780-7.

[3] Hirsa, A., Neftci, S.N., (2013). An Introduction to the Mathematics of Financial Derivatives. Publisher: Elsevier Inc. ISBN: 978-012384682-2.

[4] Hsu, M., at all, (2010). Generalized UDF for analytics inside database engine. 11th International Conference on Web-Age Information Management, WAIM 2010; Jiuzhaigou; China, ISBN: 3642142451;978-364214245-1.

[5] Huťka, V., & Peller, F. (1999). Finančná matematika v Exceli, Elita, Bratislava, 1999, ISBN 80-8044-064-6

[6] Lester, T.G., (2009). Using excel for duct calculations user defined functions. ASHRAE Journal, Volume 51, Issue 8, August 2009, Pages 42-46. ISSN:0001-2491E-ISSN:0364-9962.

[7] Microsoft: Analyze Script Performance https://docs.microsoft.com/en-us/sql/ssdt/analyze-script-performance?view=sql-server-2017

[8] Murthy, R., at all, (2003). Supporting ancillary values from user defined functions in Oracle. Nineteenth International Conference on Data Ingineering; Bangalore; India; Conference Proceeding, Pages 151-162.

[9] Mishura, Y., (2016). Financial Mathematics. Publisher: Elsevier Inc. ISBN: 978-178548046-1.

[10] Ordonez, C., at all, (2006). Vector and matrix operations programmed with UDFs in a relational DBMS. 15th ACM Conference on Information and Knowledge Management, CIKM 2006; Arlington, VA; United States. Proceedings, Pages 503-512.

[11] Ordonez, C., Garcia-Alvaro, C., (2011). A data mining system based on SQL queries and UDFs for relational databases. 20th ACM Conference on Information and Knowledge Management, CIKM'11; Glasgow; United Kingdom, ISBN: 978-145030717-8.

[12] Simhadri, V., at all (2014). Decorrelation of user defined function invocations in queries. 30th IEEE International Conference on Data Engineering, ICDE 2014; Chicago, IL; United States, ISBN: 978-147992554-4.

[13] Sousa, M., at all, (2014). Consolidation of Queries with User-Defined Functions. 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Edinburgh, Scotland. Volume: 49, Issue: 6, Pages: 554-564. DOI: 10.1145/2594291.2594305.

[14] User defined function. Available at: https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/user-defined-functions

[15] Vagač, M., Melicherčík, M., (2015). Improving image processing performance using database user-defined functions. 14th International Conference on Artificial Intelligence and Soft Computing, ICAISC 2015; Zakopane; Poland. Conference Proceeding, ISBN: 978-331919323-6.