Hybrid Technologies for Databases

Anatoly V. Anisimov, Igor O. Zavadskiy, Petro P. Kuliabko

Taras Shevchenko National University of Kyiv, Volodymyrs'ka, 64/13, Kyiv, 01601, Ukraine

Abstract

Relational approach DBTG CODASYL proposal allow us to use better features of each component. We propose some supplementary operations over the data sets – the main construction of DBTG CODASYL proposal, which can be more effective in some cases, at least due to the fact that relational databases use mostly symbolic addressing, and databases with complex structures - mostly relative. As a result, user may choose among different types of data connections concerning the speed of their processing: slow but flexible, or fast but complex.

Keywords

Hybrid technology, relational approach, relational algebra, DBTG CODASYL proposal, data set, Dribas algebra

1. Introduction

As well-known, the relational approach [1,2,3] provides the maximum simplification of the data structure to a flat file and significant increasing of the data manipulation language level, which makes it possible to create a clear and strong mathematical background for such languages.

However, the transition from more complex structures to relationships means (at least for simple queries) an impairment in processing efficiency, at least due to that relational databases use mostly symbolic addressing, and databases with complex structures - mostly relative. Nowadays many applications need to use BIG Data [7,8,9] and Cloud computing [10,11] technologies and receive the problem of data processing inefficiency. At least in part, the solution is in using different types of connections among data according to the speed of their processing: from slow but flexible to fast but complex. So, the main goal of this article is to construct an algebra (based on the relational selective Dribas algebra [2]) with the basic set that isnm the set of data sets as a supplement to the traditional algebra over relationships.

2. Selective Dribas algebra

The selective Dribas algebra was proposed a few years after the appearance of Codd's relational algebra. Relational Codd algebra is more focused on theoretical aspects, and Dribas algebra is more convenient in sense of implementation and practical use. It borrowed set-theoretic operations of Codd relational algebra (union, intersection, and difference), as well as the Cartesian operation and project. In addition, 4 filtering operations were added. The operation of β-filtering F_β (R, *condition*) is very similar to the Codd operation of Θ-restriction, but in Dribas algebra the *condition* is an arbitrary formula without quantifiers with the attributes names from the relationship R and constants, which in practice is much more convenient than in the Codd case (comparing values of different attributes, but with the same sign and are combined only by conjunction).

II International Scientific Symposium «Intelligent Solutions» IntSol-2021, September 28–30, 2021, Kyiv-Uzhhorod, Ukraine
EMAIL: anatoly.v.anisimov@gmail.com (A.1); ihorza@gmail.com (A.2); kpp1@ukr.net (A.3)
ORCID:0000-0002-1467-2006 (A.1);0000-0002-4826-5265 (A.2); 0000-0001-5411-6592 (A.3)

The next filtering operation $F_\exists$ (R1, R2, *condition*) is also similar to the Codd operation of Θ-join. The difference is that the result of this operation is a relationship, the scheme of which is a copy of the relationship R1, and not the Cartesian product of two relationships, as in the classical case. In fact, the result is only those tuples from R1, for which there is at least one tuple in R2 that satisfies the *condition*. Again, we get a practical gain compared to the classic case. The operation $F_\forall$(R1, R2, *condition*) has no direct equivalent in classical algebra, but can be expressed through other operations and was added for practical reasons. As in the previous case, the result of the scheme is the copy from R1, and is filled with those tuples from R1, for which the *condition* is equal TRUE for **all** tuples from R2. For some queries, this operation is very convenient, and its implementation largely repeats the implementation of the previous operation. The most interesting and complex in terms of execution operation F_S(R1,R2,A, $B_a\theta C$) is also a filtering operation, the result of which is some subset of tuples from the relation R1; A, B_a - lists of attributes from R1, and C - list of attributes from R2; θ is a binary predicate of set comparison: $\subseteq$, $\subset$, $\supseteq$, $\supset$, $=$, $\neq$. First, R1 is grouped by the same values of the sub-tuples from list A, then R2 is projected by the list of attributes C; then each group of tuples, more precisely sub-tuples according to the list B_a, is compared (as a sets) with R2 [C]; if the condition is TRUE, then the corresponding group of tuples from R1 passes into the result.

3. DBTG CODASYL proposal

The data set (DS) (after DBTG CODASYL [4,5]) - (as an instance) is a two-level tree with one record-owner (Owner) and several records-members (Member) of the same type. By record we mean a set of pairs {attribute name: value}, and we interpret uniformity in a relational sense: two records are considered to be of the same type if there is a bijection between their attributes that matches both the attribute names and the types of corresponding values. Thus, all records are structurally similar to tuples of relationships, which really is a simplification in comparison with the DBTG CODASYL proposal.

Two data sets are of the same type if their owner records and all member records from both data sets are of the same type. The set of the same type of data sets forms the type of data set. Some datasets have only one Owner and an empty set of Member records. There can also be records outside of any data set, then we assume that such records are a Members of some DS, the Owner of which is a special SYSTEM record. Classical relationships can be formed from the same type of records with the SYSTEM owner, over which all operations of both Codd relational algebra and Dribas relational algebra are defined. Therefore, our proposed data model can be considered as an extension of the relational model, and the data selection language - as an extension of the relational algebras of Codd and Dribas. Records may or may not have a field similar to the primary key. Database key (DBK) for written records always exists, but it can be formed on the basis of the several fields, or calculated. Although firstly it seems that the DS has less expressive power than the hierarchical or network structure, but in fact each record can be part of several different DS both as owner and as a member, forming a data structure such as "network".

4. Example in terms of ER-model

Consider an example of a subject area. First, we give its text description and ER-model (Fig. 1) [6]. There are hospitals (Hospital), there are departments (Department) in hospitals, there are chambers (Chamber) in departments, patients (Patient) reside in chambers; in addition, physicians (Physician) work in the departments; each physician is responsible for several chambers (perhaps some physician is not responsible for any chamber); physicians consult patients, one physician can consult several patients and one patient can be consulted by several physicians. Although all connections, as seen in the ER-model, have a one-to-many type, the Consulting entity actually implements a many-to-many connection between the Physician and Patient entities. We need to make one more important remark. Hospital has its number (h#), which uniquely specifies the object instance. Department also has its number (d#), but this number is unique only in one hospital (by the way such situation is quite usual for subject area with hierarchical structure), consequently in relational model we need to use for the relationship Department the pair attributes (d#, h#) as a primary key. The same situation we have for the relationship Chamber. Relationships Patient and Physician have one attribute primary key.

Now we give a description of the data domain according to the relational approach, i.e. each entity and relationship (type "many-to-many") are represented by tables with the corresponding attributes:

***Hosp**ital(h#, Hname, Hadress, rank);* ***Dep**artment(d#, h#, Dname, Dmng);*

***Ch**amber(c#, d#, h#, Cname, qnt);* ***Ph**ysician(p#, d#, h#, Pname, spec, age);*

***Pat**ient(t#, c#, d#, h#, Tname, diagn, temp);* ***Con**sulting(r#, p#, t#, data);*

Table 1

Record Description

Record Name is Hosp { H# CHAR(5), HNAME CHAR(25), Hadress CHAR(30), rank CHAR(7) };	Record Name is Dep { D# CHAR(5), DNAME CHAR(25), Dmng CHAR(30) };
Record Name is Ch{ C# CHAR(5), CNAME CHAR(25), Qnt integer };	Record Name is Ph { P# CHAR(5), PNAME CHAR(25), spec CHAR(20), age integer };
Record Name is Pat{ T# CHAR(5), TNAME CHAR(25), diag CHAR(40), temp float };	Record Name is Con { R# CHAR(5), Data DATE};

Table 2

Set Clauses

Set HD Owner is Hosp Member is Dep;	Set DC Owner is Dep Member is Ch;	Set CP Owner is Ch Member is Pat;
Set DP Owner is Dep Member is Ph;	Set PhCon Owner is Ph Member is Con;	Set PC Owner is Ph Member is Ch;
Set PCon Owner is Pat Member is Con;		

Unlike DBTG CODASYL in our case the data set as Member can have other data set that is convenient for hierarchical constructions. For example, to the already described DS we will add some more (Table 3).

5. The operations of the supplement algebra

We construct a supplement algebra based on DS, i.e. the arguments of the operations of this algebra will be DS, and the result will also be DS. Note that DS can be thought of as a function that matches each instance of Member (uniquely, but not mutually uniquely) to the instance of OWNER to which it is subordinate. Operations are focused on actions within the database, aspects of the interface will be considered separately. Data sets DS1 and DS2 must be the same type, the resulting DS will be of the same type. DS3. OWNER = DS1. OWNER U DS2. OWNER (without duplicates), Member sets are also combined without duplicates, but with subordination, i.e. if there are levels (matching DBK) records in DS1. OWNER and in DS2. OWNER, then in DS3. OWNER includes only one record, and many of their subordinate Member records are combined (without duplicates) and are subordinate to a common OWNER. This operation can cause problems: suppose that DS1 has an instance (O1, M1) and DS2 has the instance (O2, M1) then after reunion in DS3 we get the situation that M1 has two owners, which is forbidden, because it is not tree-structure; so in such situations UNION operation will leave only instance (O1, M1) in DS3.

INTERSECT(DS1, DS2) -> DS3;

Table 3
Data set as Member

Set HDC Owner is Hosp Member is DC;	Set HDP Owner is Hosp Member is DP;	Set CPCon Owner is Ch Member is PCon;
Set HCP Owner is Hosp Member is CP through DC;	Set CPCon Owner is Ch Member is PCon;	Set PhCP Owner is Ph Member is CP;

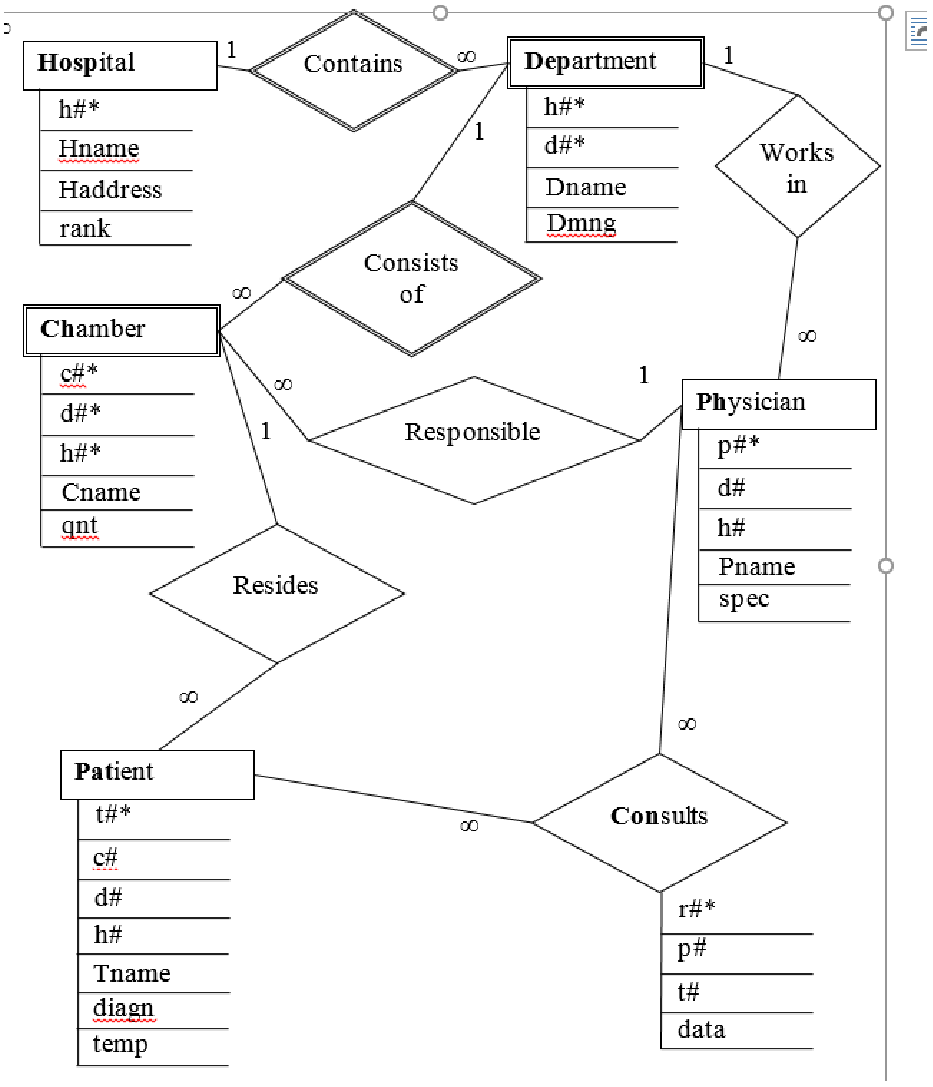

Figure 1: ER-model of the “Hospital” data domain

Data sets DS1 and DS2 must be the same type, the resulting DS will be of the same type. DS3. OWNER is an intersection DS1.OWNER ∩ DS2.OWNER, in particular it may be empty, then the entire resulting DS will be empty; Member sets also intersect, but only those for which there are instances of OWNER; instances of DS with empty sets Member can be formed.

DIFFERENCE(DS1, DS2) -> DS3;

Data sets DS1 and DS2 must be the same type, the resulting DS will be of the same type. DS3. OWNER = DS1. OWNER \ DS2. OWNER, in particular it may be empty, then the entire resulting DS will be empty; for Member subtraction is performed on subsets subordinate to one OWNER.

TIMES(DS1, DS2) -> DS3;

This operation is defined everywhere and there is no requirement on types of data sets. DS3. OWNER = DS1. OWNER $\otimes$ DS2. OWNER, instances of Member also multiply among themselves, but only within the limits of subordination to their OWNER.

PROJECT (DS1, [Olist], [Mlist]);

Olist is a list of fields for OWNER, and Mlist is a list of fields for Member. Obvious limitation: Olist must be a sublist of fields DS1.OWNER, and Mlist - a sublist of fields DS1.Member. The expression [ ] means an empty list of fields, and the expression [*] means that all OWNER or Member fields are in the list, depending on whether the expression is in the 2nd or 3rd place. In the future there will be deviations from the classical algebraic style, but this is only for more convenient use; this technique was used by E. Codd when he introduced relational algebra operations. Besides the overall case of the PROJECT operation we'll also use some special cases: PROJECT-OWNER(DS, [Olist]) -> R; and PROJECT_Member(DS, [Mlist]) -> R; For both operations their result is not DS, but relationship; if a member of a data set is a DS, the result of such operation will be a relationship (according to the list Mlist), which is a Member of the last DS.

5.1. Operations of filtration

β-filtration BFilter(DS1, condition) -> DS2;

DS2 has the same type as DS1 and is populated with data from DS1 that satisfies the condition. The condition is a formula with fields from both OWNER and Member, constants, and some unary predicates, for example: EmptyMember (NotEmptyMember) returns TRUE (FALSE) if there is at least one instance of Member for the current OWNER instance. The condition verification is valid within one instance of the data set.

ExistsFilter(DS1,DS2,condition)-> DS3;

DS3 has the same type as DS1 and is populated with data from DS1 that satisfies the *condition* if **there is at least one** such instance in DS2. The condition is defined in the same way as in the previous operation, but fields with DS2 may also be present in the formula. Condition verification is valid within one instance of DS1.

AllFilter(DS1,DS2, condition)-> DS3;

DS3 has the same type as DS1 and is populated with data from DS1 that satisfies the condition if it is executed for **all** instances of DS2. The condition is determined in the same way as in the previous operation. Condition verification is valid within one instance of DS1.

SetFilter (DS1, DS2, AOlist, BOlist θ COlist) -> DS3;

- it is the operation for set comparison. DS1, DS2 and DS3 are data sets, and the resulting data set DS3 has the same type as DS1;

AOlist and BOlist are attribute lists from DS1, and AOlist is attribute list from DS1.OWNER and BOlist is attribute list from DS1.Member; COlist - list of attributes from DS2. Member;

θ is a binary predicate of set comparison: $\subseteq$, $\subset$, $\supseteq$, $\supset$, $=$, $\neq$.

The order of actions is such:

a) a set of instances of DS1.OWNER is grouped by AOlist;

b) DS2.Member is projected by C;

c) from the groups $\{R1_{AOlist}\}$ are selected those instances for which the condition BOlist θ COlist is fulfilled;

d) into the resulting DS3 passes only those instances of DS1 that meet the specified condition.

5.2. Some other operations

JOIN(DS1, DS2, condition) -> DS3;

This operation joins two data sets DS1, DS2, which are neighbors on the hierarchical path. D (DS1. OWNER) = D (DS3. OWNER) – this means that their domains are equal; D (DS2.Member) = D (DS3. Member) and D (DS1.Member) = D (DS2.OWNER). Namely, from the 2 two-level trees is constructing three-level tree with subordination saving, then the intermediate level is removed and again we get a 2-level tree; by means of a *condition* the necessary copies are selected; the *condition* may include constants and fields of all records from DS1 and DS2.

JOIN(DS1, DS2, condition) -> DS3 ;*

Such type of operation will be appropriate when there is another DS on the hierarchical path between DS1 and DS2; its OWNER will be the same type as DS1.Member, and its Member will be the same type as DS2.OWNER, then we will get DS3, such as D(DS1.OWNER) = D(DS3.OWNER), and D(DS2.Member) = D(DS3.Member).

JOINMember (DS1, DS2, condition) -> R;

Another one specific type of connection, oriented on the situation when DS1 and DS2 have a common Member, which implements a connection between two objects of the type "many-to-many" (ER-model). The result will be a relationship that is DS1.OWNER and satisfies the *condition*.

COUNTMember(DS1) -> DS2;

For this operation DS2.OWNER = DS1.OWNER, but instead of the DS2. Member for each OWNER the corresponding quantity of elements is substituted.

COMPOSE(REC1, REC2, condition) -> DS1;

REC1 and REC2 are the records (or more exactly relationships) from which DS1 is formed, REC1 is forming the OWNER of the data set and REC2 is forming its Member. REC2 must have an attribute (or group of attributes) that is a foreign key, and for REC1 it is a primary key. Symbolic addressing between such relations will be transformed to more efficient relative. The *condition* is required to specify such attributes.

ADDMember(DS, REC, condition)

adds a REC record to the DS as a Member; *condition* is used for specification as in previous operation. This operation is convenient when we need to add some record to the data set and connect it with some particular owner; such connection is specified by *condition.*

6. Query examples

Let us consider a few examples of queries in the subject area, which was described earlier (Table 4 – Table 10). Each query will be presented in natural English, in terms of the original Dribas algebra (left column) and by means of proposed supplement algebra (right column).

7. Conclusion

The proposed extensions to classical relational algebra allow us, on the one hand, to increase the efficiency of query execution, and on the other hand, to reduce the gap between DML and SQL. From the given examples it is seen that the notation of some queries in terms of the proposed algebra looks more compact.

Table 4
Query Example 1.

Find hospital names and address for the hospitals with rank = “xxx”.	

F_β(Hosp, Rank = “xxx”) -> R1;
R1[Hname, Hadress] -> res;
	BFilter(HD, Rank = “xxx”) -> DS1;
	PROJECT-OWNER(DS1,[Hname, Hadress]) -> res;

For such simple query proposed algebra has no benefits in comparison with traditional relation algebra.	

Table 5
Query Example 2.

Find department names and hospital number in the hospitals with rank = “xxx”.	
F_β(Hosp, Rank = “xxx”) -> R1; $F_\exists$(Dep, R1,Dep.h# = R1.h#) -> R2; R2[Dep.h#, Dname] ->res;	BFilter(HD, Rank = “xxx”& NotEmptyMember) -> DS1; PROJECT (DS1, [h#], [Dname]) ->res
Supplementary algebra uses one operation less than relational algebra.	

Table 6
Query Example 3.

Find the patient names with a temperature greater than 37 from hospital № 1, department №3, chamber №6.	

F_β(Dep, h# = 1) -> R1;
$F_\exists$(Ch, R1, Ch.c# = 6 & Ch.d# = R1.d#) ->R2;
$F_\exists$(Pat, R2, Pat.temp >37 & Pat.c# = R2.c#) -> R3;
R3[Tname] -> res;
	JOIN (HD, DC, HD.H# = 1 & C# = 6) -> DS1;
	JOIN (DS1, CP, temp >37) ->DS2;
	PROJECT (DS2, [], [Tname]) -> res;
	/* or the same with DS HCP */
	BFilter(HCP, temp>37 & h#=1 & d#=3 & c#=6) ->DS2;

Relational algebra uses 4 operations, while proposed algebra uses 3 operations, even 2 if it will work through data set HCP.	

Table 7
Query Example 4.

Find the chamber names from hospital №1, where there are no patients with a temperature > 37	

(F_β(Pat, temp >37)) -> R1;
(Pat \ R1)[c#] -> R2; $F_\exists$(Ch, R2, Ch.c#=R2.c#) -> R3; R3[Cname] -> res;
	BFilter(CP, temp >37) -> DS1;
	CP \ DS1 -> DS2;
	PROJECT(DS2, [Cname], []) -> res;

Relation algebra has 4 operations while proposed algebra needs 3.	

Table 8
Query Example 5.

Find the physician names who consult at least all patients in chamber №6 of hospital №1 and department “XXX”.	

(F_β(Dep, h# =1 & Dname = «XXX»)
-> R1;
$F_\exists$ (Ch, R1, c# =6 & Ch.d# = R1.d#) -> R2;
$F_\exists$ (Pat, R2, Pat.c#=R2.c#)-> R3;
F_S(Consulting, R3, p#, Consulting.t# $\sqsupseteq$ R3.t#) -> R4;
$F_\exists$ (Ph, R4, Ph.p# = R4.p#)-> R5;
R5[Pname] -> res;
	BFilter(HD, h# =1 & Dname = XXX») -> DS1;
	JOIN(DS1, DC, c# = 6) -> DS2;
	JOIN(DS2, CP, TRUE) -> DS3;
	JOIN(DS3, PCon, TRUE) -> DS4;
	/* JOIN*(DS2,PCon,TRUE) -> DS4;
	Join over the hierarchical path */
	SetFilter(PhCon, DS4, Ph.p#, PhCon.Con.r# $\sqsupseteq$ DS4.Con.t#) ->DS5;
	PROJECT-OWNER(DS5, [PNAME]) ->res;

This query is classified as complex because it needs to perform the set comparison.	

Table 9
Query Example 6.

Find the patient names consulted by Dr. Sidorchuk.	

F_β(Ph, Pname = " Sidorchuk ") -> R1;
$F_\exists$(Con,R1,Con.p#=R1.p#)->R2;
$F_\exists$(Pat,R2, Pat.t#=R2.t#) ->R3;
R3[Tname] -> res;
	JOINMember(PCon,PhCon, PCon.Con.r# = PhCon.Con.r# & PhCon.Ph.Pname = " Sidorchuk ") -> R1;
	R1[Tname] -> res;

This query is performed on the base of connection of "many-to-many" type.	

Table 10
Query Example 7.

Find the patient names who are consulted *only* by those physicians who consulted the patient "xxx" by name.	

(F_β(Pat, Tname = "xxx"))[t#] -> R1;
$F_\exists$(Con,R1,Con.t#=R1.t#)->R2;
R2[p#] -> R3; Con[p#,t#] -> R4;
F_s (R4, R3, t#, R4.p# $\subseteq$ R2.p#) -> R5;
$F_\exists$(Pat,R5, Pat.t#=R5.t#) ->R3;
R3[Tname] -> res;
	Join*(PatPh, PhPat) -> DS1;
	OnlyFilter(DS1,Member.Tname="xxx")

This query is performed on the base of connection of "many-to-many" type. OnlyFilter operation is the special case of the SetFilter operation; it is equivalent to SetFilter (DS1, DS1.Owner, DS1.Member, $\subseteq$, *condition*). Its running algorithm may be the follows: firstly we mark such member-records for which the *condition* is FALSE and then for each owner-record its subordinate records are verified while marked one is appeared. If it will be no marked records among the subordinate records then such owner-record should be added to the result.	

8. References

[1] Codd, E.F. A Relational Model of Data for Large Shared Data Banks // Communications of the ACM: journal, 1970, Vol. 13, no. 6, P. 377—387, doi:10.1145/362384.362685

[2] Dribas V.P. Relational models of databases.// Publ. BGU. Minsk. 1982, 192 p.

[3] Date C.J. An Introduction to Database Systems 6th edn. Reading, MA: Adison-Wesley. 1995, 1328 p.

[4] Olle T.W., The CODASYL approach to data base management, A.Wiley-Interscience Publication, John Wiley & Sons, 1978.

[5] CODASYL Database Task Group Report. (1971). ACM, New York, April 1971, 487 p.

[6] Chen P.P. The Entity-Relationship model – Toward a unified view of data. ACM Trans. Database Systems, 1(1), 1976, 9-36.

[7] Building a robust, governed data lake for AI, IBM Hybrid Data Management, https://www.ibm.com/downloads/cas/RMAMZNRY (Nov. 2021)

[8] Botelho B., Bigelow S.J., Big Data, (Nov. 2021), https://searchdatamanagement.techtarget.com/definition/big-data

[9] Tlamelo Emmanuel, Thabiso Maupong, Dimane Mpoeleng, Thabo Semong, Banyatsang Mphago and Oteng Tabona, A survey on missing data in machine learning, Journal of Big Data 8, N140 (2021), https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00516-9

[10] Sai Vennam, Cloud Computing, (18 August 2020), https://www.ibm.com/cloud/learn/cloud-computing

[11] Steve Ranger, An introduction to cloud computing right from the basics up to IaaS and PaaS, hybrid, public, and private cloud, AWS and Azure. https://www.zdnet.com/article/what-is-cloud-computing-everything-you-need-to-know-about-the-cloud/ (Nov.2021)