Schema Optimisation Instead Of (Local) Normalisation

Bernhard Thalheim[0000−0002−7909−7786]

Christian-Albrechts University at Kiel, Department of Computer Science, D-24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de
http://www.is.informatik.uni-kiel.de/∼thalheim

Abstract. Classical normalisation theory has a number of lacunas although it is commonly and widely accepted and it is the basis for database theory since the 80ies. Most textbooks and monographs still follow this approach despite the good number of open problems. Today, modern object-relational DBMS offer far better capabilities than the systems that have been built in the past based on the strict relational paradigm. Constraint maintenance has been oriented on transformation of structures to structures that are free of functional dependencies beside key constraints. The maintenance of coherence constraints such as two-type inclusion constraints has been neglected although this maintenance might be the most expensive one. In reality normalisation is local optimisation that exclusively considers functional dependency maintenance.
We thus need a different normalisation approach. This paper develops an approach towards optimisation of schemata and global normalisation. This approach results in a denormalisation and object-relational database schemata.

Keywords: normalisation · schema optimisation · denormalisation · performance · object-relational databases · global normalisation

1 Normalisation - The Good, The Bad, The Ugly

Normalisation is considered to be one of the pearls of database theory. There is almost no database course that does not teach this part of a theory. The main results have been achieved during the 70ies, 80ies and early 90ies. Since then the theory is considered to be completed although new DBMS (database management systems) and new database paradigms have been developed since then. There are very few publications on object-relational structures. XML approaches have mainly be following this research paradigm.

1.1 Local Vertical Normalisation Based on Functional and Other Dependencies

Local database normalization aims at the derivation of database structures that can easily be supported by the DBMS. In the past, DBMS supported keys, domain constraints and referenced-key-based inclusion constraints (so-called foreign-key-constraint). Therefore, it was a goal to derive another equivalent schema to the given one which has a set of integrity constraints that can be supported by the DBMS used for implementation. This approach can be understood as a descriptive approach to optimisation of database

structuring depending on the platform for implementation. Normalisation as a concept is typically too narrow and too much focussed on local vertical normalisation.

Three kinds of normalisation. Normalisation is mainly considered to be *vertical normalisation* on the basis of projection sets as mappings and join as the restoration operation. *Horizontal normalisation* [23] is based on selection and union. *Deductive normalisation* [30] is based on reduction of classes according to tuple-generating constraints and extended completion using this set of tuple-generating constraints (see, for instance, [3]). It is the most storage effective and the best computational method for normalisation as long as the tuple-generating dependency used for decomposition is acyclic [30, 31]. The latter two normalization methods have not yet got a proper support by the database systems vendors. A common treatment for these three kinds has not yet developed.

The consideration of horizontal normalisation together with vertical normalisation has a number of advantages. A type may be first decomposed horizontally what would enable application of another vertical normalisation to one of the horizontal subclasses[1]. Additionally, horizontal normalisation enables in separation of a class into data that are stable and will not be changed and volatile data that are still changed. The first data class can then be supported by a large set of indexes and improve performance.

Normalisation theory is a relict of the 80ies. The relational database design book by [41] is one of the most comprehensive surveys on kinds of normal forms. It essentially considers almost 30 kinds. Some of them have been superseded by other, some of them did not find practical solutions or their specific kinds of applications.

Normalisation starts with the requirement of the first normal form, i.e. all attributes are atomic and do not have an inner structure. Consider, however, the *ZIP* as an attribute of an *address*. It has an inner structure which is guilty for the non-BCNF normalisation of addresses [21]. Meanwhile DBMS suppport user-defined data structures (UDT's). These data structures have their specific functions and predicates. The *address* example is an academic one while in practice address data are checked against public address databases. A similar observation can be made on the second normal form and others. They can be neatly supported by modern DBMS. The support goes far beyond what non-first-normal-form (NF^2) research provides.

Beside the flaws of the synthesis algorithm for normalisation discussed below we should pay attention whether the *collection step* (step 3 in normalisation: collect all FD's with the same left side into a singleton new relation type) is really appropriate. Some of the FD's may have a different flavour. Consider, for instance, an *organisation unit* which is *chaired by somebody*, has its *postal address*, has a main *financial funding*, a secretariat or *office*, and a *contact*. Why it should be represented as a singleton type? It is far better to use a star kind structure since each of the associations has its meaning and its usage in the database application. Such structures are also causing evolution problems.

Normalisation approaches typically do not consider the different meanings of the dependencies [35] but treat the set of all constraints under consideration as a set of

[1] This mechanism has been already used for fragmentation techniques for distributed databases. A similar approach has been proposed by C.J. Date [10] for handling NULL-polluted classes by rigid horizontal normalisation into NULL-free fragments (see also [40, 25]).

elements which are of equal importance. Moreover, normalisation is based on classes of constraints such as functional dependencies and multivalued dependencies. As we shall see below, normalisation should however been based on sets of constraints from different classes and on the meaning of such constraint sets.

Another obstacle of normalisation theory is the assumption that relation classes are sets. SQL allows multi-sets. The normalisation theory for multi-sets has a different setting [18].

1.2 Local Vertical Normalisation

Local vertical normalisation plays a prominent role in normalisation theory. Research started with the introduction of FD-based normal forms (first, second, third in a good variety of notions, Boyce-Codd) and led to a good body of knowledge[2] Later functional dependencies have been generalised to domain, multi-valued, and hierarchical dependencies and the generalisation of the last one to join dependencies. This research resulted in introduction of further normal forms, e.g. fourth and fifth. The sixth normal form (6NF) has already been introduced with the DBMS MIMER in the mid-70ies.

Local normalisation is a good approach as long as we restrict the consideration to strictly equality-generation dependencies such as functional dependencies and singleton relational schemata with atomic attributes. In this case the so-called third normal form is achievable. The Boyce/Codd Normal Form (BCNF) is not achievable in any case. [21] has shown however that all known counterexamples are based on ill-defined structures. It can be shown [31] that either hierarchical decomposition or refined granularity of attributes result in BCNF structures.

Reasons to normalise. [31] surveys the main targets of normalisation: (1) avoiding inadequate behaviour such as anomalies, (2) elimination of unnecessary redundancy, (3) prohibiting inconsistent data, (4) stability of database schemata during application evolution, (5) optimising database performance, and (6) maintenance of abstraction levels within a schema. We refer to [31] for discussion of other problems encountered for normalisation such as adequate BCNF representation, adequacy of decomposed schemata, competing normalisations of the same schema, and inadequacy of multivalued dependencies and other tuple-generating dependencies within the relational database model.

We observe that the first three reasons are rather operational one whereas the last two are tactical ones. The fourth reason is a strategic one that is the main source for later modernisation, migration, and re-engineering.

Did we achieve the six targets? The answer is no. We achieved the first target and partially achieved the second target. The third target can only be achieved if all potential constraints and their influence on consistency is handled properly. The fourth target has not yet found good research solutions. After evolution schemata suffer from mannerism and look similar to Gothic cathedrals or chaotic sandcastles. After normalisation, database performance might be good for data manipulation operations. Normalisation

[2] We restrict the citations to the most essential ones for this paper and restrain to give a full survey of the research.

might result in a far worse behaviour for database querying. The advent of data warehouses is a reaction on this problem. The sixth target is not supported by current languages that force us to stay on one abstraction level.

1.3 Inclusion Constraint Maintenance After Decomposition

Literature often neglects the set of additional inclusion constraints that must be maintained after a decomposition of a class. Given a multivalued dependency $X \twoheadrightarrow Y$ for a partition X, Y, Z of the set of attribute of a relation type R. The class R^C can be decomposed into $R_1^C = \pi_{X \cup Y}(R^C)$ and $R_2^C = \pi_{X \cup Z}(R^C)$ in such a way that $R^C = R_1^C \bowtie R_2^C$ iff the multivalued dependency is valid in R^C. We note that this multivalued dependency is implied by a functional dependency $X \rightarrow Y$.

This vertical decomposition of R^C into R_1^C and R_2^C must be maintained by pairwise inclusion dependencies $\pi_X(R_1) \subseteq \pi_X(R_2)$ and $\pi_X(R_2) \subseteq \pi_X(R_1)$. In the relational DBMS setting the pairwise inclusion constraint should be maintained by foreign key constraints, i.e. X should be a key in both R_1^C and R_2^C.

Question 1. Is there any good approach to inclusion constraint maintenance after decomposition?

A solution to this foreign key requirement is proposed in the RM/V2 model [9] by introduction of a third relation type $R_0^C = \pi_X(R^C)$.

Observation 1. This approach results in a number of additional auxiliary relation types what limits the effect of normalisation. [3]

We need to add to the normalisation approach also an *extended dependency preservation rule* that is often neglected in the literature:

Principle 1. The decomposition based on vertical normalisation adds to the decomposed types pairwise inclusion dependencies on intersecting attributes. The decomposition based on horizontal normalisation adds to the decomposed types an exclusion constraint.

This principle has already been implicitly used for the universal relation assumption. We observe however that pairwise inclusion dependencies may cause severe performance problems.

The union constraint for horizontal decomposition is implicit and is the basis for defining a view that combines by UNION the decomposed components into the original type[4]. The deductive normalisation [31] is another option.

1.4 Constraint Sets Instead of Sets of Constraints

The classical approach of computer science introduces syntax first. And then semantics is defined on top of syntax. As again discussed in [35], this approach is nice for computational handling and for inductive and incremental construction but completely

[3] We forbear from postulating these observations as theorems. They are rather simple and easy to check statements.

[4] We note that a database schema is typically not a database model. The schema must be enhanced by views to become a database model [36]. Since we have to use anyway views then we should better extensively use horizontal decomposition beside vertical decomposition.

unnatural for normal languages. Syntax, semantics, and pragmatics form a unit. The *syntax-semantics-separation* principle finds its rigid continuation in the separation of integrity constraints into classes that have some uniformity in their definition structure. This separation principle has been found well-acceptable by programmers and logicians. It is however completely counterintuitive [35]. Natural language use a holistic semiotic approach and do not artificially separate units that form a semiotic holistic statement. Additionally, constraints might have their own meaning [28] such as syntactic functional dependencies compared to dependencies that represent semantical units.

The main deficiency is the constraint acquisition problem. Since we need a treatment for sets a more sophisticated reasoning theory is required. One good candidate is visual or graphical reasoning that goes far beyond logical reasoning [12].

Star and snowflake structures used in OLAP approaches are the basis for an approach that handles structures as a complex within its structure and its semantics in the complex. With the advent of object-oriented and XML languages we learned lessons on object identification [2] and the co-use of set-based classes with pointers. These approaches can be considered as a starting point.

Let us extend the open problem (TIC7) (Real-life constraint sets [33]):

Problem 1. Provide a reasoning facility for treatment of heterogeneous sets of constraints instead of constraints from a given constraint class. Classify 'real life' constraint sets which can be easily maintained and specified.

In [13] we realised that the classical Hilbert-type reasoning (premises allow to derive a conclusion) should be replaced by another schema: some premises which are supported by other constraints allow to derive a conclusion. This set-of-support reasoning can be based on graphical reasoning means or spreadsheet reasoning schemata.

1.5 The Storyline of the Paper

Several reasons can be observed why local normalisation may be inadequate. We shall discuss some of them in the next Section. We restrict the discussion to relational database technology and to conceptualisation through the extended entity-relationship model [31]. Some of the (88 [sic!]) pitfalls of object orientation [39] and of XML orientation have similar causes but are outside the scope of this paper. Normalisation theory is so far exclusively built as a theory of vertical local normalisation. We might ask whether we should consider global vertical normalisation. Or at least other kinds of local normalisation as well. The main target of normalisation of optimisation of the overall database for all six targets. Instead of poly-optimisation for some of the six criteria we might use a less strict form by optimisation of some of the database types and by denormalising others.

Since poly-optimisation is typically unsolvable we develop a number of corrections to normalisation approaches and a general approach to denormalisation as a kernel for a general theory of optimisation.

2 Solutions for Classical Normalisation Approaches

2.1 Refining Synthesis Algorithms

Rigidity of classical synthesis algorithms. The third step of the classical synthesis algorithm typically groups all attributes that can be inferred from the same set of attributes by functional dependencies. This approach groups then attributes that are potentially conceptually completely independent into one group. An alternative approach could be rigid non-grouping, i.e. the left hand side of a functional dependency $X \longrightarrow Y$ is the basis of k new types with attributes $X \cup B_i$ for $Y = \{B_1, ..., B_k\},\ 1 \leq i \leq k$. Both approaches are extreme positions. We may observe, however, that some separation must be maintained.

Let us consider a simple example [35] of a relational type R: given attributes
$attr(R) = \{A, B, D, F, G, I\}$ and a set of functional dependencies
$\Sigma_R = \{A \longrightarrow IG, D \longrightarrow FG, IAB \longrightarrow D, IF \longrightarrow AG\}$.
This FD set can be represented by the graph on the left side of Figure 1. This set can be reduced by deleting $IF \longrightarrow G$ from the graph since it is derivable through the edges representing $IF \longrightarrow A$ and $A \longrightarrow G$. Furthermore, the set ABI can be reduced since the edge representing $A \longrightarrow I$ already supports subset reduction. No other reduction can be applied to the graph. We use the calculus for graphical reasoning [12] that is complete and sound. We use dotted lines for the subset relationship among subsets of attributes and arrows for functional dependencies.

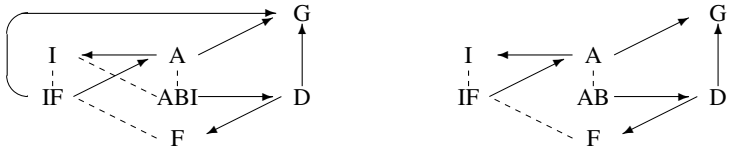

Fig. 1. The graph of the functional dependencies and the reduced cover of this set

We may directly derive a normalisation according to this graph reduction. Each constraint must be covered. We arrive with the synthesis algorithm to[5]:
$R_1 = (\{A, G, I\}\ ,\ \{A \longrightarrow GI,\ R_1[AI]\ \subseteq \supseteq\ R_2[AI]\})$,
$R_2 = (\{A, F, I\}\ ,\ \{A \longrightarrow I, FI \longrightarrow A,\ R_2[F]\ \subseteq \supseteq\ R_4[F]\})$,
$R_3 = (\{A, B, D\}\ ,\ \{AB \longrightarrow D,\ R_3[D]\ \subseteq \supseteq\ R_4[D],\ R_1[A]\ \subseteq \supseteq\ R_3[A]\})$,
$R_4 = (\{D, F, G\}\ ,\ \{D \longrightarrow FG,\ R_1[G]\ \subseteq \supseteq\ R_4[G]\})$.
The set $\{A, B\}$ is a key. We thus do not need an additional key type for the normalisation.

If we however take into account constraint maintenance and redundancy then we arrive at a smaller and better schema with the type:

[5] If we require that all inclusion dependencies are referential integrity constraints then we need 12 types for normalisation that results in a foreign-key-faithful decomposition: $R_1[\underline{A}], R_1[\underline{I}], R_1[\underline{A}, I], R_1[\underline{A}, G], R_2[\underline{F}], R_2[A, \underline{I, F}], R_3[\underline{B}], R_3[\underline{A, B}], R_3[\underline{A, B}, D]$, $R_4[\underline{D}], R_4[\underline{G}], R_4[\underline{D}, F, G]$ where the key of each new type $R_i[X] := \pi_X[R_i]$ is underlined.

$R_1' = (\{A, G\}\ ,\ \{A \longrightarrow G,\ R_1[A]\ \subseteq\supseteq\ R_2[A]\})$
due to the validity of the following derivation (reduction rule):

$$\frac{R_1[A,I] \subseteq\supseteq R_2[A,I],\ R_2 : A \to I,\ R_1 : A \to GI}{R_1 \bowtie R_2\ = R_1' \bowtie R_2,\ R_1' : A \to G}\ R_1' = \pi_{A,G}(R_1) \qquad .$$

This rule is based on general deduction rules and on equalities for the relational algebra:

Theorem 1 (*General deduction for closed Horn formulas*). *A sound and complete axiomatisation for closed Horn formulas* $\forall ...(\alpha \to \beta)$ *consists of*

axioms

$$\frac{}{\alpha \to \beta} \qquad \text{for all facets of substructures } \beta \preceq \alpha$$

,

augmentation rules for super-structures α^+ ***and sub-structures*** β^- $\frac{\alpha \to \beta}{\alpha^+ \to \beta^-}$
for either $\beta^- \preceq \beta$ *and* $\alpha \preceq \alpha^+$ *or as well as* $\alpha^+ = \alpha \sqcup \gamma$ *and* $\beta^- = \beta \sqcap \gamma$ *,*
and

transitivity rules $\frac{\alpha \to \beta,\ \beta \to \gamma}{\alpha \to \gamma}$ *for all connecting pairs* ($\alpha \to \beta$,$\beta \to \gamma$).

The proof of the theorem is based on the Boolean representation of the open first-order predicate calculus and on properties of implications. The completeness uses the same arguments as the classical FD completeness proof.

An alternative proof of the reduction rule is based on algebraic dependencies [23, 30].

We observe that graphical synthesis would result in a better behaviour. This structure is represented in Figure 2. The hyper-network approach [24, 34] uses nodes as Venn diagrams of subsets of the set of attributes and directed edges between the nodes. The constraint set in Figure 1 is given by the hyper-network representation in Figure 2. Compare this representation to the decomposition hypergraph for the classical synthesis algorithm [17]. The hyper-network representation is rather simple to read. Each of the edges must be represented by some new relation type. Moreover, the set $\{A, B\}$ is a key due to the graph node closure. Otherwise we may use a combination of nodes for the graph node closure. The second minimal key is $\{I, F, B\}$ which is not a node and thus would have been added to the decomposition if we would not have represented the first one. We notice that graphical reasoning is simpler for implications than Hilbert-type calculi, e.g. [3, 14, 40].

Questions one might ask for normalisation theory and their research agenda. The classical normalisation theory is based on functional and multi-valued dependencies. Normalisation synthesis algorithms are deterministic. The result depends on the order of attributes and on the order of constraints considered in the algorithm. The minimal cover is not unique for a given set of constraints (even not polynomial according to the number of attributes in the worse case). We, thus, have a good number of opportunities for a normalisation.

Question 2. Which normalisation opportunity should be the best one?

The solution cannot be to consider only one of them. We might used a pragmatistic solution however: choose the most performing one and keep the knowledge on alternatives.

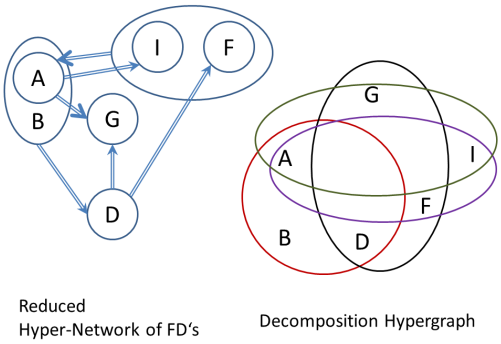

Fig. 2. Graphical normalisation depicted as hyper-network and decomposition hypergraph.

The classical approach is based on hypergraphs. Instead we should use hyper-networks and meta hyper-networks [35] (see also Fig. 2).

Let us consider a very simple set Σ of functional dependencies for
$R = (\{A, B, C, D, E, F, G, H\}, \Sigma)$ with
$\Sigma = \{A \to B \to C \to AD, D \to E \to F \to DG, G \to H \to G\}$.
Σ has more than 50 minimal covers. Similar examples can be given for results of normalisation according to synthesis algorithms.

Observation 2. Attribute sets which are FD-equivalent can be given in an abstract form, i.e. we consider in a set of constraints the complex $[A, B]$ *instead of* $\{A \to B, B \to A\}$.

The attribute-wise consideration might be appropriate for first normal form definitions but it complicates reasoning on constraints.

We thus represent Σ by the rather simple FD set system $\{[A, B, C] \to [D, E, F] \to [G, H]\}$. It has 18 different BCNF normalisations (similar case in [16, 17].

Question 3. Why we should consider so many minimal covers and normal forms?

We note that multivalued dependencies are defined in the relational database theory in a mathematical manner. They are far better expressed by entity-relationship modelling languages [32] and far simpler to capture and to develop.

Question 4. Should we better develop a normalisation approach for entity-relationship schemata? Should we better consider a schema for normalisation instead of type-wise normalisation?

Good HERM schemata are typically the best normalisation. Folklore of ER modelling claims that the best normalisation is obtained if the main target of conceptual modelling is the specification of an ER schema. This claim is not valid in general since extended entity-relationship modelling languages such as HERM [31] are not cognitively complete. The ER approach provides however a far better solution to normalisation of relational schemata than normalisation theory, e.g. for multivalued and hierarchical dependencies. Moreover, the structure is more natural and thus better to comprehend.

A flaw of the first normalisation algorithms was corrected by a *key composition rule*, i.e. if the decomposed relational structure does not have a constructed type which contains some key of the old schema as a (sub-)structure then a new type is created for one of the minimal keys and added to this relational structure. This rule is nothing else as a decomposition for which a relationship type is added with a key that has the property that it is overwriting the key product of related decomposed types. The corresponding hyper-network has then nodes which are not incrementally layered and thus need a connecting element which can be then used as a key. The hyper-network approach also allows a generalised and less strict key composition rule.

2.2 Balancing Between Conceptualisation and Programming Adequacy

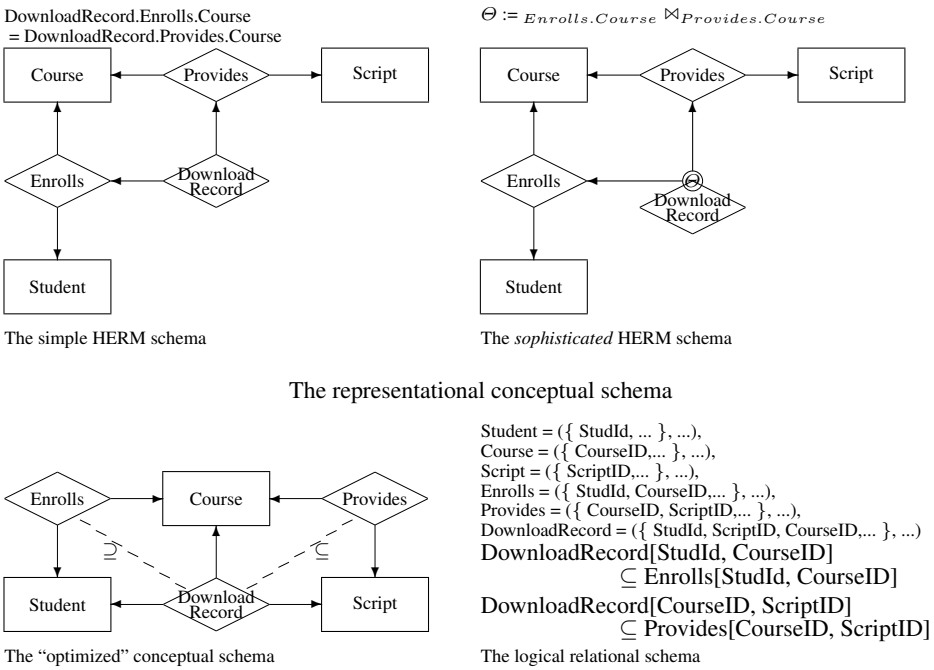

Fig. 3. The 'Janus' schema cluster for conceptual modelling

Database design and development is often based on the three-layer architecture. This architecture requires that the user can be supported by views defined on top of the conceptual schema. The conceptual schema is mapped to the logical and to the physical schemata. The last two are considered to be internal or implementation schemata. It is well known in database design [27, 29] that the conceptual schema be represented by a 'Janus' schema, i.e. one schema (*representational conceptual schema*) that is used for

conceptual representation, for view formulation and for conceptual programming and another schema (*optimized conceptual schema*) that is used for implementation issues including logical and physical programming. The two schemata are equivalent to each other and are tightly associated with each other by transformation mappings. A typical example of these two schemata is given in Figure 3. The example is taken from a script server project. Students enrolled in a course may download scripts that are provided by the course.

The optimised conceptual schema can be easily mapped to a structure that supports smooth operating of the database. We can deduct from this schema the internal representation, supporting structures such as indexes (in various kinds), internal concatenation or splitting of relations, introduction of generalisations, the navigational structure for access, the management of derived structures (derived attributes, views, macro-data on top of the micro-data of the database), criteria for the selection of internal storage and computational structures, the necessity for the introduction of artificial or surrogate keys, and for clustering or separation of records. These parameters are used for tuning and physical optimisation of the database. The sophisticated HERM schema uses the Θ-join for the correct building of the relationship type that records downloads. The optimised conceptual schema is equivalent to this schema due to the equivalence of the join decomposition and the inclusion constraints [31].

2.3 Accuracy of the Internal Database Structure

The internal database structure is ruled by the DBMS. The mappings from the conceptual schema to the internal schema must preserve a number of properties:

Preservation of content: The conceptual schema has been established by modelling the application domain and highlights data that are going to be stored in a database. The internal schema represents the same and only the same data that can be recorded through the conceptual schema and is based on the database modelling language of the platform assumed for implementation.

Preservation of access and modification: The access pathes and modification pathes that are implicitly or explicitly assumed for the conceptual schema are mapped to access and modification pathes that are entirely supported by the platform assumed for the implementation.

Preservation of maintenance efficiency: Integrity constraints are given in a declarative form in conceptual schemata. The efficiency of their maintenance is not considered. The (economic) value of constraints is neglected. Internal schemata must provide mechanisms for efficient integrity maintenance.

The first property is an element of any database course and well considered in most database books. The property can be treated on the basis of non-losslessness and dependency preservation. It might be enhanced by the requirement that each projection (or other range structure) should be needed for the reconstruction process. The second property is mainly solved by the professional experience of database professionals. It is typically not discussed in scientific publications and is an element of database operator education. The third property is often neglected in database research. Database operators have a very simple solution for this property: they switch off all those integrity

preserving processes that become a bottleneck at database production time and switch on these constraints for a short time at the maintenance phase.

Preservation of access and modification is a fuzzy criterion since we often may not assume that any access and modification can be forecasted at design time. The co-design approach to database modelling [31] also takes into consideration functionality that can be envisioned. Optimisation of the internal schemata is based on profiles of functions, stored procedures and transactions (shortly processes) and on the cardinality profile of the relations. The first profile provides information on the kind of the operation, the frequency, the relations affected by the processes, the type of access (online, batch, prefetch or ad-hoc) and the changes applied to the databases. The cardinality profile of relations provides detailed information on the size of the relation (minimal, maximal and average), on the changes of the size over time and the associations among relations that must be maintained. Both profiles are compared with the modus of computation (bath, online, ad-hoc), with performance expectations (execution time, throughput, priority), with visibility of performance gaps (depending on operations frequency, organisation level of users, business process interaction) and with computation strategies for the operations (kind of operation, scheduling, auxiliary facilities, selection and storage alternatives, set sizes).

Decomposition approaches generate structures that easily support some of the constraints such as key constraints, domain constraints and key-based inclusion constraints. Typically, the maintenance complexity of such constraint sets is not taken into account. Moreover, decomposition algorithms may generate a large variety of decompositions that are semantically equivalent but pragmatically and technologically different. Typical normalisation algorithms are deterministic for a given set of functional dependencies, for an order of attributes and an order of the dependencies. Changes in the the last two orders result in different solutions of those algorithms.

2.4 Infomorphisms among Schemata

We use the notion of infomorphisms as the general foundation for schema optimisation. Infomorphisms have been used for schema modernisation in [15, 38]. A typical example of an infomorphism is the association of a relational database schema and a sophisticated XML schema (with a good number of additional constraints since XML uses list (or tree) and reference types instead of values). The relational database schema that is obtained by the classical forgetful mapping from an entity-relationship schema is not an infomorphism since the ER structuring is richer than the relational structuring.

Let us consider two *database* ***schemata*** $\mathcal{S}_1$ and $\mathcal{S}_2$ consisting of *database types* of the form $T = (struc(T), \Sigma, \Sigma^*)$ with a structure definition, inner integrity constraints Σ defined on the type, and outer integrity constraints Σ^* that constrain the type by means of other types. Structure elements of types that are not defined by constructors are called *basic*.

Let us consider only *complete schemata*, i.e. those which types are complete relative to the outer constraints. Given furthermore, *basic domain types* $\mathcal{B}$ for the value foundation of the database. We use the abstract data type approach for basic domain types and presume for these types their value collections, their operations, and their predicates.

An *extended database schema* $\mathfrak{D} = (\mathcal{S}, \mathcal{B}, DOM)$ consists of a database schema and an assignment DOM of basic elements of its types to basic domain types.

The set of all $MOD((\mathcal{S}, \mathcal{B}, DOM))$ of all finite databases on $\mathfrak{M}$ consists of finite collections of classes for each type for which all constraints are valid, which values of objects in a class are given by DOM.

Let us now associate databases for different extended database schemata $\mathfrak{D}_1$ and $\mathfrak{D}_2$ by mappings mappings $\widehat{put}_{1,2}$ and $\widehat{put}_{2,1}$. These two mappings form an ***infomorphism*** of $MOD(\mathcal{D}_1)$ and $MOD(\mathcal{D}_2)$ if for i, j with $\{i, j\} = \{1, 2\}, i \neq j$ and for each database DB_i on $MOD(\mathcal{D}_i)$ there exists a database DB_j and on $MOD(\mathcal{D}_j)$ such that $\widehat{put}_{i,j}(DB_i) = DB_j$ from one side and $\widehat{put}_{j,i}(DB_j) = DB_i$ from the other side.

We may extend this notion also to views defined on each of the database schemata. The association among views can be based on the extract-transform-load (ETL) approach where extraction is based on a query language of the first schema, transformation is given by an infomorphism, and loading uses views which allow updates on the second database schema.

This notion is very general one. Infomorphisms are essentially transformations of one database to another one. These transformations are *information-invariant* in the sense that any database object collection can be associated with one and only one database object collection from the other extended database schema.

The infomorphism notion can be based on HERM schema operations ([31], Chapter 9.2.) in the case that we consider only classes with set semantics. The Σ^* dependence among types also includes inclusion constraints. Therefore, vertical normalisation can be directly expressed in this approach. In this case, we can represent transformations as graph-grammar rules which are defined on sub-schemata. Horizontal normalisation uses separating selection σ_{α_i} predicates which define a partition of singleton classes. Deductive normal forms use for the mapping a reduction operation from one side and a chase-like completion procedure for the second mapping.

Observation 3. Vertical normalisation, horizontal normalisation, and deductive normalisation are specific variants of infomorphisms.

We conclude now that a theory of infomorphisms can subsume the classical relational normalisation theory, especially vertical normalisation. It is, moreover, better since the pairwise inclusion constraints after decomposition must be integrated into the decomposed schema. Infomorphisms can be partially supported by schema construction rules for extended entity-relationship schemata. These rules follow the graph grammar approach.

2.5 Global and Local Vertical Normalisation

The synthesis algorithm is also based on *structure minimality*, i.e. the type structures form a Sperner set in the sense that $struct(T_1) \not\sqsubseteq struct(T_2)$ is not valid for any two types T_1 and T_2 of a schema $\mathcal{S}$. Structure minimality reduces the maintenance. It might, however, provide its advantages as we already illustrated for the RM/V2 approach. Moreover, additional structures such as overlapping and subtype indexes (as hedges of indexes) may support performance of computation and also input-output to a real essential extent.

Global normalisation concurrently and coherently considers all types of a database schema. The example in [31] allows to derive five different results of a normalisation of a small schema. Each of these schemata have their advantage.

We may define a result of a normalisation process that is applied to an entity-relationship schema as a type-wise transformation of the given schema by an infomorphism, i.e. the types of a schema are (vertically or horizontally) decomposed to a schema in which all types are in certain α-normal form ($\alpha \in \{1, 2, 3, 4, 5, 6, BCNF\}$). Since decomposed types may be a substructure of another one, we use these types only once. Entity, cluster, and relationship types are transformed by graph grammar rules [31]. The decomposition of a relationship type follows the procedure developed in [22].

A schema $\mathcal{S}$ is a *global α-normal form* schema if all its types are in α-normal form and if the schema is structure-minimal. Within a platform setting $\mathfrak{P}$, we add the requirement that all its integrity constraints can be supported by declarative means provided by the platform. Otherwise, the schema is called *$(\alpha,\mathfrak{P})$-unnormalised*.

Observation 4. Global normalisation is based on an infomorphism.

3 Denormalisation

> *... There are many database experts, particularly the more academic ones, who feel that any talk about denormalising a database is like a race car driving – a great way to mangle or kill yourself even if you know what your are doing.* [8]

3.1 State-Of-the-Art for Denormalisation

We observe two camps where the first one is well acknowledged.

No denormalisation at all! Almost all[6] textbooks and monographs in the database area require strict normalisation. Local (vertical) normalisation of a singleton database type is well reflected in most database books (e.g. [1, 5, 19, 41]) and publications, most database courses, and in actual database practice. It is considered as one of the pearls of database research and known to almost everybody who knows database technology. The provenance and acknowledgement is based on the facility it provides: keeping as much as possible locally and globally supporting only those processes that are inherently global. Both independence concepts of databases (conceptual independence and implementation independence) are based on localisation. [11] advocates lazy normalisation based on relevant and quickly to capture FD's, i.e. somehow ***liberal normalisation***[7] for which not all functional dependencies that are valid in database schema are considered but only the really important ones[8].

[6] We know so far only less than a handful books that do not require such.

[7] Many constraints can be omitted since integrity is also often managed through proper interfacing and exchange procedures without a chance for inconsistency as long as the data modification is exclusively based on interface or exchange view data. The development of a theory for this approach is one of the lacunas of database theory.

[8] We avoid the exponential size ***trap*** for sets of functional dependencies with this toleration of incompleteness of constraint sets. We consider only essential ones and completely or partially neglect others. This approach can be extended to a *theory of robust normalisation*.

Additionally, almost valid FD's might be more important than FD's that happens to be valid. The treatment of such dependencies would be based on the introduction of artificial identifiers, i.e. a heavy object identity pollution. A far better solution is horizontal decomposition with a class for which all identities are valid and an exception class in which the few exceptions are recorded. Horizontal decomposition can be combined with union views as long as the exceptions are disjoint from the normal case.

Liberal and controlled denormalisation whenever it is really necessary: Very few papers and books advocate or consider at least to some extent denormalisation (e.g. [6–8] or the discussion in [31]). The three central quality criteria for database installations are, however, performance, performance, and performance. The classical vertical local normalisation is useful as long as any casual user may query by any casual query at any time without considering performance. However, a database system contains of a (or a number of) DBMS with a number of databases on top of which a large massive of business procedures has been developed. These business procedures form the main part of the profile of the database. Casual queries are rather exceptions. The definition that is used for denormalisation is typically based on application of the natural join operator to relational types[9].

Our consulting experience and also observations on the why's for OLAP and data warehouse applications drives us to a completely different picture in many applications. The first setting of a database application is very often based on normalisation. This database becomes then step by step denormalised after the database is fully populated and operating. Already after one year of full operation, the database is partially normalised and also partially denormalised.

3.2 A Matter of Definition

Denormalisation has not yet been defined in the literature despite [4]. Essentially, we find two approaches (e.g. in [6, 8, 10] for simple forms):

Denormalisation as the inverse of normalisation: Given a schema with α-normalised types. Any non-trivial combination (typically by a join operation) of two or more types that defines an infomorphism is a denormalisation.

Denormalisation as the extension of a schema: Given a schema $\mathcal{S}$. Any extension of the schema that is defined by an infomorphism is called denormalisation.

Typical extension operations are [8]: prejoined types for types with complex queries, reports added to the schema, mirrored types, type splitting by horizontal decomposition, partial combination of types, introduction of controlled redundancy for data, repeating groups, hierarchy tables, and overloading of types.

Given a $(\alpha,\mathfrak{P})$-normalised schema. Any infomorphism transformation of this schema to an unnormalised one is called ***denormalisation***. Index and other supporting means thus do not change the normalisation status of a schema.

We may consider at the same time normalised and denormalised schemata. A theory and techniques for denormalisation for physical schemata based on normalised conceptual (or logical) schemata have been developed in [29]. [20] lists some key effects of

[9] The validity of pairwise inclusion constraints is also neglected in this case.

thoughtful denormalisation: definite improvement in query time, a potential increase in update time or in storage space, a potential loss of data integrity due to certain deletions, the necessity for program transformations for all relevant queries and the overhead needed to reorganise some tables. Strict local normalisation may be inadequate. Denormalisation may result in a more complex maintenance complexity. It may also lead to complications for query formulation. It becomes easier to formulate incorrectly a query to a request meaning that the query does not correspond to the request. Often tricky view creation is used for denormalised tables. The denormalisation is considered a method for performance improvement despite discussed so far advantages of normalisation.

Our definition of denormalisation does not allow composition by equi-join since we have to avoid the NULL marker problem. NULL markers must be treated depending on their specific meaning, their occurrence, and their impact on computation. We however support co-existence of vertical and horizontal normalisation and denormalisation.

Therefore, it seems that normalisation is the best way for optimisation of database behaviour. A theory of denormalisation has not yet been proposed as far as we know.
Question 5. What are the denormalisation criteria? Is there any theory for it? Is there any 'playground' approach for consideration of (de)normalisation?

Instead, a number of heuristic rules for denormalisation are provided. These rules are based on observations for performance traps for some of the platforms and often use the 80/20% rule.

3.3 Denormalisation Driven By Optimisation

We base our approach on essentials of database performance forecasting, tuning techniques, and database distribution into fragments [4, 26]. Our approach has been implemented in an industrial setting and for performance improvement for a very large cluster of databases [4, 37].

Let us first define the performance portfolio of a database application and the profile of a DBMS. A *portfolio* consists of a set or collection of tasks. A *profile* of a DBMS specifies the services and the capability of a DBMS. The *extended database application schema* consists of the database schema, the business processes and the characterisation of the application demand by a *characterisation of the kind of computation* based on the description of the operations involved, the operation support, and the data volumina transferred for support of computation, the *visibility description of processes* for the business user that includes frequency of operations and their relation to business processes, the *description of the modes of computation* such as online, batch and interactive mode of computation or deferrable and immediate application of computation, the *performance properties and quality* based on the expected execution time for online etc. modes, based on the throughput expectation for queries, modifications and transactions, based on restrictions such as suitability or response time, and based on priority claims issued by the business user, the *criticality level* of the processes.

We derive now the measures for this application, a database schema, and a DBMS:

Data modification costs: Given a set M of modification operations m_i with their weight wm_i in the application, the frequency of application hm_i of each operation, their

complexity of realisations m_i^* in the DBMS $\mathcal{P}$, and the complexity of integrity maintenance sm_i for the operation m_i.
The complexity m_i^* can be computed type-wise for types T from the schema $\mathcal{S}$, i.e. by m_{iT}^*.
The modification complexity $modify(\mathcal{S}, \mathcal{P}, M)$ is given by the formula:

$$\sum_{m_i \in M} ((\sum_T (m_{iT}^* + sm_i) \times wm_i) \times hm_i)$$

Query cost: Given a set Q of queries q_j with their weight wq_j in the application, the frequency of application hq_j of the query q_j, the complexity q_j^* of the realisation of q_j, and the complexity of integrity query imposed integrity maintenance sq_j for the query q_j.
The complexities can be computed type-wise for the types T from the schema $\mathcal{S}$, i.e. by q_{iT}^*.
The query complexity $querymodify(\mathcal{S}, \mathcal{P}, Q)$ is given by the formula:

$$\sum_{q_j \in Q} ((\sum_T (q_{jT}^* + sq_j) \times wq_j) \times hq_j)$$

The **schema complexity** $complexity(\mathcal{S}, \mathcal{P}, M, Q)$ is the sum of the modification complexity and of the query complexity.

An infomorphism can be now extended to the modification and to the query operations under consideration of the base types and the domain assignments $(\mathcal{B}, DOM)$.

We can now compare the complexity of the schema according to the modification and the query portfolio of a given application.

Given two extended database schemata $\mathfrak{D}_1 = (\mathcal{S}_1, \mathcal{B}_1, DOM_1)$ and $\mathfrak{D}_2 = (\mathcal{S}_2, \mathcal{B}_2, DOM_2)$ and an infomorphism $(\widehat{put}_{1,2}, \widehat{put}_{2,1})$ for these two schemata; further, given a platform $\mathcal{P}$, and a modification and query portfolio.

The extended database schema $\mathfrak{D}_1$ performs better than the database schema $\mathfrak{D}_2$ in a given setting $\mathcal{P}$ for a portfolio $M \cup Q$ if

$$complexity(\mathcal{S}_1, \mathcal{P}, M, Q) \ << \ complexity(\mathcal{S}_2, \mathcal{P}, M, Q) \qquad .$$

We use $<<$ as a denotation for an essential discrepancy of the two complexities.

We may thus derive the normalisation and denormalisation criterion for schema optimisation for given schemata where $\mathcal{S}_1$ is $(\alpha, \mathcal{P})$-normalised and $\mathcal{S}_2$ is $(\alpha, \mathcal{P})$-denormalised:

Use the normalised schema $\mathcal{S}_1$ if $\mathcal{S}_1$ performs better than $\mathcal{S}_2$ in the given setting.
Use the denormalised schema $\mathcal{S}_2$ if $\mathcal{S}_2$ performs better than $\mathcal{S}_1$ in the given setting.

We neglect within this approach the existence of casual queries and of casual data modification. This approach, however, supports typical applications where the retrieval and also the modification is well-defined at the development orr at later maintenance time.

Observation 5. The optimisation approach to normalisation and denormalisation allows to coherently meet demands for the six reasons why we should normalise.

4 Conclusion

4.1 Summarising

Normalisation is considered to be one of the pearls of database theory and technology. We mainly consider, however, local vertical normalisation instead of global normalisation or horizontal normalisation. It seems that normalisation theory is a body of knowledge that is completely settled and well understood in most of its aspects. We discuss on the basis of simple examples that this impression is not valid. Normalisation is not well understood. It needs a lot of extensions and corrections. It must also be completely revised for the modern DBMS technology. One essential revision is the flexible choice for set-based or multi-set-based semantics. This extension opens the path towards list, pointer, multi-list, etc. semantics that is supported nowadays by systems.

Normalisation is often a performance bottleneck. Repairing this bottleneck is often done on the fly. In practice, skilled consultancy uses here a hands-on, experience-backed approach. The DAMA[10] community and database forums widely discuss in closed groups the experience some people got. We claim that most larger database applications allow coexistence of partial normalisation (in both vertical and horizontal style) and partial denormalisation.

This paper aims now to highlight the path to a coherent theoretical underpinning for this kind of coexistence. We first discussed problems of classical normalisation based on the verticality and locality approach for simple constraints such as functional and multivalued dependencies. The problems discussed can be resolved by pragmatical approaches. Some of them are discussed in the paper. We are not capable to present a full theory which would require a two-volume monograph. So, we restricted only on some parts of this theory.

Normalisation can be understood as a special kind of optimisation. As such it should be treated as "a commandment" [10] unless the database application requires high query performance. Optimisation of schemata is based on some kind of equivalence. We use infomorphisms as one solution for treatment of equivalence. This solution requires deep knowledge of the the given database application. It can be extended to handling of robust constraint sets what is, however, an open issue. It can also be extended to handling by basic-structure normalisation that is neatly supported by interface and exchange tolerance as long as the interfaces and the exchange means provide a support for the other optimisation (or more specifically normalisation) requirements.

4.2 Open Problems

The list of open problems is slowly shrinking and quickly expanding at the same time. We have collected open problems since MFDBS'87[11], have extended this list, and observed whether some of them have been resolved. The latest version in [33] contains 22 open problems which are directly related to normalisation theory.

[10] www.dama.org

[11] With 21 open problems from which 13 are not yet solved.

Normalisation theory is currently a theory for system structures in the small. Global normalisation will be theory for system structures in the large. The world is now changing to systems in the web and systems that are based on completely different performance challenges such as big data massives. *Normalisation in the world* is a really big issue for future research. It goes far beyond theories we know for distributed databases.

Revolution instead of unworthy extension. Already research on the OO identifier and the OID pollution has been demonstrating that parts and pieces of database theory must be revised. Many assumptions taken for granted are not valid anymore and will never be valid again for challenging applications such as big data massives. Set semantics was a nice tool in the past. It is not the right one - at least for SQL applications and multi-sets in practice. we might ask why not also to use multi-list semantics. Big data requires a different FD logic.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, Reading, MA, 1995.
2. C. Beeri and B. Thalheim. Identification as a primitive of database models. In *Proc. FoMLaDO'98*, pages 19–36. Kluwer, London, 1999.
3. A. A. Benczúr, A. Kiss, and T. Markus. On a general class of data dependencies in the relational model and its implication problems. *Computers & Mathematics with Applications*, 21(1):1 – 11, 1991.
4. M. Bick. Denormalisierung. Master's thesis, CAU Kiel, Dept. of Computer Science, 2015.
5. J. Biskup. *Foundations of information systems*. Vieweg, Wiesbaden, 1995. In German.
6. S. Buxton, L. Fryman, R. H. Güting, T. A. Halpin, J. L. Harrington, W. H. Inmon, S. Lightstone, J. Melton, T. Morgan, T. P. Nadeau, B. O'Neil, E. J. O'Neil, P. E. O'Neil, M. Schneider, G. Simsion, T. J. Teorey, and G. Witt. *Database Design - Know It All*. Morgan Kaufmann, 2008.
7. J. Celko. *Joe Celko's SQL for smarties - Advanced SQL programming*. Morgan Kaufmann, San Francisco, 1995.
8. J. Celko. *Joe Celko's Data and Databases: Concepts in Practice*. Morgan Kaufmann, 1999.
9. E. F. Codd. *The relational model for database management (version 2)*. Addison-Wesley, Reading, MA, 1991.
10. C. J. Date. *Database Design and Relational Theory - Normal Forms and All That Jazz*. O'Reilly, 2012.
11. C.J. Date. *Go Faster – The transRelational approach to DBMS implementation*. C.J. Date & Ventus Publishing ApS, 2011.
12. J. Demetrovics, A. Molnar, and B. Thalheim. Graphical and spreadsheet reasoning for sets of functional dependencies. In *Proc. ER'2004*, LNCS 3255, pages 54–66, 2004.
13. J. Demetrovics, A. Molnar, and B. Thalheim. Graphical and spreadsheet reasoning for sets of functional dependencies. Technical Report 0402, Kiel University, Computer Science Institute, http://www.informatik.uni-kiel.de/reports/2004/0402.html, 2004.
14. A. Kiss and T. Markus. Functional and inclusion dependencies and their implication problems. In *10th Int. Sem. on DBMS*, pages 31–38, Cedzyna, Poland, 1987.
15. M. Klettke and B. Thalheim. Evolution and migration of information systems. In *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 12, pages 381–420. Springer, Berlin, 2011.

16. H. Köhler. Autonomous sets - A method for hypergraph decomposition with applications in database theory. In *FoIKS 2008*, volume 4932 of *Lecture Notes in Computer Science*, pages 78–95. Springer, 2008.
17. H. Köhler. Autonomous sets for the hypergraph of all canonical covers. *Ann. Math. Artif. Intell.*, 63(3-4):257–285, 2011.
18. H. Köhler and S. Link. SQL schema design: foundations, normal forms, and normalization. *Information Systems*, 76:88–113, 2018.
19. M. Leonard. *Database design theory*. MacMillan, Houndsmills, 1992.
20. S. Lightstone, T. Teorey, and T. Nadeau. *Physical database design*. Morgan Kaufmann, 2007.
21. J. A. Makowsky and E. V. Ravve. Dependency preserving refinements and the fundamental problem of database design. *DKE*, 24(3):277–312, 1998. Special Issue: ER'96 (ed. B. Thalheim).
22. H. Mannila and K.-J. Räihä. *The design of relational databases*. Addison-Wesley, Wokingham, England, 1992.
23. J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The structure of the relational database model*. Springer, Berlin, 1989.
24. G.P. Popkov and V.K. Popkov. A system of distributed data processing (In Russian). *Vestnik Buryatskogo Gosudarstvennogo Universiteta*, (9):174–181, 2013.
25. K.-D. Schewe and B. Thalheim. NULL value algebras and logics. In *Information Modelling and Knowledge Bases*, volume XXII, pages 354–367. IOS Press, 2011.
26. D. E. Shasha and P. Bonnet. *Database Tuning - Principles, Experiments, and Troubleshooting Techniques*. Elsevier, 2002.
27. G. Simsion and G.C. Witt. *Data modeling essentials*. Morgan Kaufmann, San Francisco, 2005.
28. O. Sörensen and B. Thalheim. Semantics and pragmatics of integrity constraints. In *SDKB'11, LNCS 7693*, pages 1–17. Springer, 2013.
29. M. Steeg. *RADD/raddstar - A rule-based database schema compiler, evaluator, and optimizer*. PhD thesis, BTU Cottbus, Computer Science Institute, Cottbus, October 2000.
30. B. Thalheim. *Dependencies in relational databases*. Teubner, Leipzig, 1991.
31. B. Thalheim. *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin, 2000.
32. B. Thalheim. Conceptual treatment of multivalued dependencies. In *ER'2003*, LNCS 2813, pages 363–375, 2003.
33. B. Thalheim. Open problems of information systems research and technology. In *Invited Keynote, BIR'2013. LNBIB 158*, pages 10–18. Springer, 2013.
34. B. Thalheim. Conceptual models and their foundations. In *Proc. MEDI2019, LNCS 11815*, pages 123–139. Springer, 2019.
35. B. Thalheim. Semiotics in databases, keynote paper. In *Proc. MEDI2019, LNCS 11815*, pages 3–19. Springer, 2019.
36. B. Thalheim and M. Tropmann-Frick. The conception of the conceptual database model. In *ER 2015*, LNCS 9381, pages 603–611, Berlin, 2015. Springer.
37. M. Tropmann and B. Thalheim. Performance forecasting for performance critical huge databases. In *Proc. EJC 2010*, pages 214–233, Jyväskylä, 2010.
38. Q. Wang and B. Thalheim. Data migration: A theoretical perspective. *DKE*, 87:260–278, 2013.
39. B. F. Webster. *Pitfalls of object-oriented development: a guide for the wary and entusiastic*. M&T books, New York, 1995.
40. Z. Wei and S. Link. Embedded functional dependencies and data-completeness tailored database design. *PVLDB*, 12(11):1458–1470, 2019.
41. C.-C. Yang. *Relational Databases*. Prentice-Hall, Englewood Cliffs, 1986.