Achievements of Relational Database Schema Design Theory Revisited

Joachim Biskup

Fachbereich Informatik,
Universität Dortmund,
D-44221 Dortmund,
Germany,
biskup@ls6.informatik.uni-dortmund.de

Abstract. Database schema design is seen as to decide on formats for time-varying instances, on rules for supporting inferences and on semantic constraints. Schema design aims at both faithful formalization of the application and optimization at design time. It is guided by four heuristics: Separation of Aspects, Separation of Specializations, Inferential Completeness and Unique Flavor. A theory of schema design is to investigate these heuristics and to provide insight into how syntactic properties of schemas are related to worthwhile semantic properties, how desirable syntactic properties can be decided or achieved algorithmically, and how the syntactic properties determine costs of storage, queries and updates. Some well-known achievements of design theory for relational databases are reviewed: normal forms, view support, deciding implications of semantic constraints, acyclicity, design algorithms removing forbidden substructures.

1 Introduction

Due to its great importance for database applications, database schema design has attracted a lot of researchers, and, accordingly, a lot of insight into good schemas has been obtained. On the one side, practical experience suggests to follow some basic design heuristics, which have been ramified into considerable detail. On the other side, theoretical investigations have accumulated many formal notions and theorems on database schema design. Unfortunately, however, theory apparently does not have much impact on practice yet.

The purpose of this paper is to improve on this mismatch of theory and practice by presenting well-known theoretical results on schema design within a fresh and unifying framework. In a companion paper [Bis95a] we also discuss the present shortcomings of database schema design theory and suggest some directions for its future elaboration. This paper does not aim at providing a complete survey on well-established and current contributions but at highlighting important examples. Accordingly, all references to the literature are to be understood just as hints for further reading.

2 Problem of Schema Design

The purpose of a database system can be roughly summarized as follows: a database system aims at persistently and dependably storing a large amount of *structured* data, shared by *many and various* users, and at *efficiently* managing this data with respect to update execution and query evaluation. The database itself, i.e. the structured data, is organized in a self-describing way: it consists of a time independent part, its *schema*, and a time-varying part, its *instance*, where the schema describes the structure and the formal semantics of the possible instances. At design time, the database administrator (representing the group of people involved), basically, has to perform two steps: first abstracting and modeling the application, then formalizing and formatting the model.

In the first step, the administrator *models* the application at hand by employing some well-disciplined linguistic framework for descriptions of reality, let's say the framework of the widely accepted entity-relationship approach [Che76]. In the second step, the administrator *formalizes* the model and declares a database schema, using a data definition language, which is an implemented fragment of a first order logic, esssentially.

This two-step procedure is based on the fundamental paradigm of the semantic triade: reality – space of concepts (ideas) and laws of logic – language. Fig.1 sketches the basic assumptions of this paradigm, which is not at all obvious or indisputable, but rather a tried attempt helpful for restricted tasks.

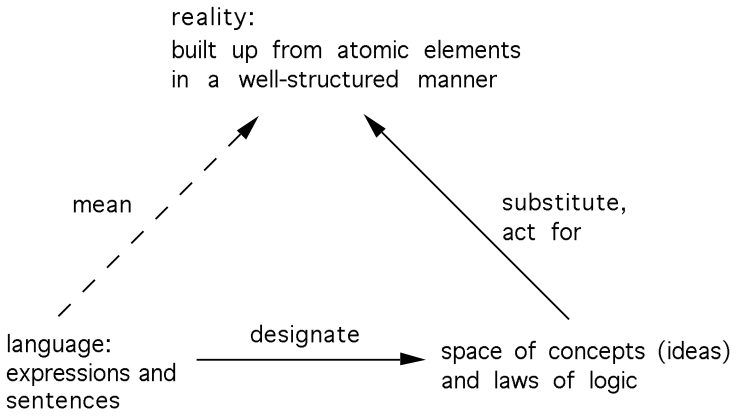


Fig.1. Fundamental paradigm of the semantic triade

Fig.2 indicates some correspondences between the framework of the entity-relationship approach, an approved heuristic tool for detecting and reconstructing the pertinent concepts, and the syntax of first order logic and its set theory based semantics, a well-studied formal language for declarative programming.

ER-approach	*logic and set theory*	
	syntax	*semantics*
entity { simple / composed	constant symbol / ground term (with function symbol)	element of a universe
relationship	ground fact	tuple of a relation
property (attribute)	{ (binary) ground fact / ground term	tuple of a (binary) relation / value of a function
role	place of a predicate symbol	component of a relation
abstraction		
universe of discourse	set of constant symbols	universe
separation/specialization	formula Φ	comprehension[1], power set
generalization	$\vee$	union
aggregation	$\wedge$, $=$	intersection, Cartesian product
constraint	(implicational) statement	model class
key constraint	with equality-conclusion	
isa constraint		
partition constraint		
many-one constraint	with equality-conclusion	
existence constraint		
referential constraint		
view rule	set of formulas	relation
action		
message	statement	
(positive) information	conjunctively added statement	reduction of model class

Fig.2. Some correspondences between the entity-relationship approach and first order logic with set theory based semantics

Both steps of the design require taking decisions. In the first step the administrator has to decide on the relevance of certain aspects of the "miniworld" under consideration. However, it is important to realize that a database system can be seen under three different though related viewpoints:

- the system constitutes a formal image of an outside miniworld;
- or it manages an autonomous formal miniworld (of documents, for example);
- or it mediates formal messages between communicating actors (one actor inserts a message, which is later on delivered to another actor as a query result, for example).

[1] If M is a structure with universe d and Φ is a formula with free occurances of the variables $x_1, \ldots, x_n$, then the *comprehension* is defined by

$d_{M,\Phi} := \{(\beta(x_1), \ldots, \beta(x_n)) \mid \beta$ is variable assignment into universe d, and Φ is true in structure M under assignment $\beta\} \subset \bigtimes_{i=1,\ldots,n} d.$

In fact, the last viewpoint appears to be most comprehensive:

- human individuals act communicatively within the (outside) miniworld (of the application),
- the basic facts and events of which are reflected by formal documents
- that in turn are mediated over time and space by the database system.

Thereby the database becomes part of the already overwhelming "formalism reality" [Bis94] surrounding its users.

Once the decisions about the relevant aspects are available from the first step of the design, in the second step the administrator has to decide on the structure of their formalization. More specifically, he has to decide on the following problems, essentially:

- Which aspects of the application should be *enumerated*, i.e. represented by a time-varying enumeration of ground facts the *formats* of which are statically declared in the schema?
- Which aspects of the application should be *inferrable*, i.e. derivable from the time-varying enumerations, possibly complemented by additional input, by *rules* which are declared in the schema?
- Which aspects should *constrain* the enumerations under updates, i.e. which format-conforming enumerations of ground facts should be considered meaningful in the sense that they satisfy *semantic constraints*, which are declared in the schema.

The decisions result in a *schema* that comprises

- the *formats* for enumerations (the time-varying extensional instances produced over the life time of the database),
- the *rules* (for intensional views supporting queries),
- and the *semantic constraints*.

Being fixed over the time, the schema statically determines the future dynamic behaviour of the database and, in particular, its usefulness for its end users. Fig.3 illustrates the design and the usage of a database, and it summarizes the terminology introduced so far.

The quality of a schema can be evaluated along two lines of reasoning:

- The schema should *formalize* the application as *faithful* as achievable.
- The schema should allow to execute queries and updates, as far as these operations can be foreseen, as efficiently as possible. From this point of view, schema design can be understood as *optimization at design time*.

Whether a *faithful formalization* has been achieved or not, cannot be evaluated solely based on formal mathematical reasoning. Rather we have to investigate whether the database will successfully provide technical support for communications among those persons that employ the database as end users. Presumably, successful support is based on a common agreement on the following questions:

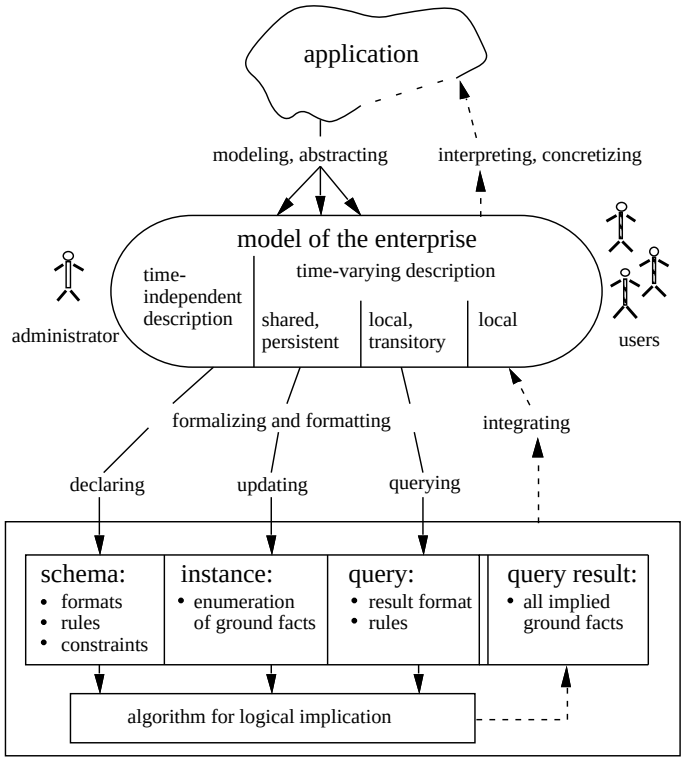


Fig.3. Design and usage of a database

- Which *entities* are to be considered basic?
- Which *relationships* are to be considered basic and
- how to select from the basic ones those for actual redundancy-free enumerations, such that all relationships can be completely inferred?
- Which actions are to be considered basic?

Optimization at design time, however, can be evaluated in formal mathematical terms by considering

- storage costs (basically determined by the size of the enumerated instances),
- query costs (basically the time complexity of anticipated queries, in particular those that are declared as rules in the schema),
- update costs (basically the time complexity of anticipated insertions and deletions, including maintenance of the semantic constraints that are declared in the schema).

3 Design Heuristics

Most guidelines for schema design can be summarized by the following four heuristics:

Separation of Aspects: A declared format should be appropriate to enumerate *exactly one aspect*.

Separation of Specializations: A declared format should be appropriate to conform to *exactly one specialization* of an aspect.

Inferential Completeness: All meaningful aspects that are not enumerated according to a declared format should be inferrable by using the query language.

Unique Flavor: Meaningful aspects should be identified and understood by expressing their basic attributes only (and omitting additional context information).

Clearly, an administrator will tentatively apply a separation heuristic by using an agreement that the entities or relationships of some class are considered basic. But afterwards he has to justify that property mathematically with respect to the formally declared schema and the inferential power of the formal query language. Similarly, an administrator will tentatively apply the Inferential Completeness heuristic by using an agreement on the selection of basic relationships for enumerations, and afterwards he has to justify the claimed completeness property mathematically with respect to the selected formalization. In the same spirit, an application of the Unique Flavor heuristic is, firstly, based on some intuitive agreements, which, afterwards, are subject to mathematical verification with respect to the selected formalization.

Having in mind the achievements of design theory presented in the rest of this paper, we will somehow artificially distinguish desirable syntactic properties of schemas from worthwhile semantic requirements: the former properties only refer to the purely syntactically given schema, whereas the latter requirements are explicitly related to the semantics of the query language. Accordingly, the separation heuristics will primarily suggest desirable syntactic properties, and the completeness and uniqueness heuristics worthwhile semantic requirements. It should be understood, however, that, on the one side, syntax and semantics are always closely related, and, on the other side, in computing we aim at eventually finding appropriate syntactic expressions for any kind of notion.

4 Tasks of Design Theory

In order to be helpful in achieving faithful formalizations and in pursuing the design heuristics, the following tasks of design *theory* are due:

- **Task 1:** Formalize the worthwhile semantic requirements and the desirable syntactic properties of schemas!

- **Task 2:** State and prove relationships between the formalized versions of worthwhile semantic requirements and desirable syntactic properties!
- **Task 3:** Find algorithms for deciding on or even achieving syntactic properties of schemas, and prove their correctness and efficiency!

In order to be helpful for optimization at design time, additionally, design *theory* should tackle a fourth task:

- **Task 4:** Prove that desirable syntactic properties actually ensure low costs!

Being supplied with appropriate solutions for these tasks, an administrator can effectively benefit from design theory. For, at design time,

- the administrator, essentially, has to deal with syntactic material only (supported by Task 3)
- which must be evaluated with respect to its semantic properties (as stated by Task 1 and Task 2) on the one side
- and the future operational cost (as stated by Task 4) on the other side.

5 Achievements for Relational Databases

5.1 Notations

For the sake of readability and conciseness we will employ (more or less) standard notations in a somehow sloppy, and sometimes also imprecise, way. In order to study carefully elaborated versions of the notations and of the results, the reader should consult the references, in particular the textbooks [Mai83, Ull88, Ull89, PDGvG89, Vos91, MR92, AD93, AHV95, Bis95b].

(R_i, X_i, SC_i)[1]	denotes a relation scheme where
R_i	is a relation symbol,
X_i	is a set of attributes (possibly with a range for its values), i.e. a format, and
SC_i	are the local semantic constraints.

A (database) schema comprises relation schemes, rules, and global semantic constraints:

$\langle(R_1, X_1, SC_1), \ldots, (R_n, X_n, SC_n)\mid$	relation schemes for extensional enumerations,
$Q_1, \ldots, Q_m \mid$	rules (queries) for intensional views,
$SC_{global}\rangle$[1]	global semantic constraints.

Semantic constraints are denoted as follows where X, Y, Y_i, Z are sets of attributes and R_i, R_j are relation symbols:

[1] Later on we will sometimes omit those components which are not relevant for the current discussion. For instance, using the notation $(\ , U, SC)$, we indicate that only the set of attributes U and the semantic constraints SC are important, but not the omitted relation symbol.

$X \rightarrow Y$	functional dependency,
$X \twoheadrightarrow Y \mid Z$ or $\bowtie [X \cup Y, X \cup Z]$	multivalued dependency,
$\bowtie [Y_1, \ldots, Y_k]$	join dependency,
$\Pi_X(R_i) \subset \Pi_Y(R_j)$	inclusion dependency,
SC^+	implicational closure of a set of semantic constraints SC.

5.2 Normal Forms: Separation of Aspects Formalized as Desirable Syntactic Property

The first design heuristic, Separation of Aspects, can be rephrased by considering formats and semantic constraints as some kind of structure and by requiring that any nontrivial substructure should correspond to, refer to or identify exactly one aspect of the application. Depending on the class of semantic constraints involved, we can define different notations of "nontrivial substructure"; but in all cases the notion of "exactly one aspect" is related to the concept of identification of unit pieces of information. In order to formalize the heuristic as desirable syntactic property, normally referred to as "normal form", see Task 1, we favor expressing the separation requirement in a negative form: the structure should *not* contain any *forbidden substructures* that might be harmful with respect to the quality measures. Then most algorithms to achieve high quality schemas can be conveniently described as iterated *schema transformations* that stepwise detect and remove forbidden substructures.

The most popular normal forms are listed in Fig.4, giving their names and forbidden substructures [Cod70, Cod72, Fag77, Del78, Zan76, BBG78, Fag81, Ken83, MR86, BDLM91, DF92]:

name	forbidden substructures
3 NF, third normal form	$Z \rightarrow A \in SC^+$, $A \notin Z$, A nonkey-attribute, (but) $Z \rightarrow X_i \notin SC^+$.
BCNF, Boyce/Codd normal form	$Z \rightarrow A \in SC^+$, $A \notin Z$, (but) $Z \rightarrow X_i \notin SC^+$.
4 NF, fourth normal form	$X \twoheadrightarrow Y \in SC^+$, $Y \not\subset X$, $X \cup Y \subsetneqq X_i$, (but) $X \rightarrow X_i \notin SC^+$.
5 NF, fifth normal form	$\bowtie [Y_1 \ldots Y_k] \in SC^+$, $\bowtie [Y_1 \ldots Y_{i-1}, Y_{i+1}, \ldots, Y_k] \notin SC^+$ for $i = 1, \ldots, k$, (but) there exists j : $Y_j \rightarrow X_i \notin SC^+$.
referential normal form	$\Pi_X(R_i) \subset \Pi_Y(R_j) \in SC^+$, $i \neq j$, (but) $Y \rightarrow X_j \notin SC^+$.
unique key normal form	$X \rightarrow X_i \in SC^+$, X minimal, $Y \rightarrow X_i \in SC^+$, Y minimal, (but) $X \neq Y$.

Fig.4. Normal forms and their forbidden substructures

5.3 View Support: Inferential Completeness Formalized as Worthwhile Semantic Requirements

The third design heuristic, Inferential Completeness, can be rephrased by considering those aspects of the application that are not explicitly represented by enumerations and by requiring that these aspects are completely supported as intensional views by appropriate rules. There are, essentially, three versions of support: *view instance support*, *view query support*, *view update support*.

Restricting our discussion to one-relation views or even so-called universal relation views, we suppose that a database schema of the form
$DS = \langle$schemes for extensional enumerations$|$ $|$global semantic constraints$\rangle$
is given, and that some candidate view (or external schema)
$ES = (\ , U, SC)$
with set of attributes U and semantic constraints SC should be supported.
Then we state the following formal versions of the heuristic as worthwhile semantic property, see Task 1.

- Schema DS provides *view instance support* for ES
  :iff there exists a query Q on DS such that
  $\{instances\ of\ ES\} \subset Q[\{instances\ of\ DS\}]$.

If we have even equality, the view instance support is called *faithful*. In that case, if, additionally, the supporting query Q is injective on $\{instances\ of\ DS\}$, the view instance support is called *unique*.

- Schema DS provides *view query support* for ES
  :iff for each query P on ES there exists a query P' on DS such that:
  for all instances u of ES there exists an instance $(r_i)_{i=1,\ldots,n}$ of DS such that
  $P(u) = P'((r_i)_{i=1,\ldots,n})$.

Under some rather weak assumptions on the query language we have a fundamental equivalence [AABM82, Hul86, BR88]:

Theorem 1. *DS provides view instance support for ES*
iff DS provides view query support for ES.

If DS provides view query support for ES, then the query P' corresponding to the identity query on ES supports the instances of ES. On the other hand, if DS provides view instance support for ES by some query Q, then Q can be composed with queries on ES. Such compositions yield a query translation from queries on the view to queries on the full schema.

For the support of *updates* on views, however, we essentially need that the view instance support is *unique*. For otherwise, well-known as the *view update problem* [BS81, DB82, FC85, Kel86, GHLM93], there is no information available to resolve the ambiguity caused by non-injectivity.

5.4 Syntactic Characterization of View Support

According to Task 2, the worthwhile semantic requirements of view support should be related to desirable syntactic properties of a schema. The main results available concern universal relation views, the supporting query of which is the natural join. For instance we have the following theorems [Ris77, Ris82, BBG78].

Theorem 2. *A schema DS with formats $X_1, \ldots, X_n$ for the extensional enumerations (ignoring local and global semantic constaints of DS)* supports *a universal relation view $(\ , U, SC)$ by* the natural join
iff $\bowtie [X_1, \ldots X_n] \in SC^+$.

Theorem 3. *A schema DS with formats $X_1, \ldots, X_n$ for the extensional enumerations and functional dependencies $F_1, \ldots, F_n$ as local semantic constraints* faithfully supports *a universal relation view $(\ , U, F)$, where F is a set of functional dependencies,* by the natural join
if $\bowtie [X_1, \ldots X_n] \in F^+$ *and* $(\bigcup_{i=1,\ldots,n} F_i)^+ \supset F$.

The proof of Theorem 2 is straightforward just by confirming that the formal semantics of join dependencies is appropriately defined. The faithfulness of the natural join results from the inclusion $r_j \supset \pi_{X_j}(\bowtie_{i=1,\ldots,n} r_i)$, showing that functional dependencies that are valid in some component r_j are also valid in the join $\bowtie_{i=1,\ldots,n} r_i$. More refined results appear for example in [Var82, CM87].

In [Heg94] a rather general theory of schema decomposition is presented. This theory explores an algebraic framework, in which the class of instances of a database schema is partially ordered and possesses a least element and the inverses of supporting queries (which are the projections in case of a natural join) are isotonic and preserve least elements. It turns out that, within this framework, the components of a schema DS faithfully supporting a universal relation view uniquely "complement" each other. Besides treating many further topics, the theory also deals with the union as supporting query (with the selection as inverse) and thus with the so-called horizontal decomposition [DP84, PDGvG89], and it clarifies the role of null values in schema decomposition.

5.5 Deciding Desirable Syntactic Properties for Normal Forms and View Support

Both heuristics treated so far finally lead to syntactic properties that are basically expressed in terms of implications of semantic constraints. In Section 5.2 normal forms, formalizing the Separation of Aspects heuristic, are just defined in these terms, and in Section 5.4 view support, formalizing the Inferential Completeness heuristic, has been reduced to these terms. According to Task 3, then, we have to design algorithms to decide implications among semantic constraints and, additionally, to explore all relevant implications systematically .

Here are some prominent examples for results [Arm74, Men79, Bis80, BV84a,

BV84b, Var84, Mit83, KCV83, CFP84, CV85, FV84, Var88a, Tha91, BC91, Her95]:

Theorem 4. *The implication problem of "$\Phi \in SC^+$" is decidable for "many important classes" of semantic constraints.*

Theorem 5. *The implication problem "$\Phi \in SC^+$" is undecidable for the class of "functional and inclusion dependencies".*

The important semantic constraints can be expressed as implicational first order logic formulae. Then the various proof procedures, called chase procedures, are based on specialized versions of the more general proof techniques of (hyper-) resolution and paramodulation. Roughly described, hyperresolution, applied to an implicational formula with a nonequality-conclusion and its previously generated premises, yields the conclusion as additional statement, and paramodulation, applied to an implicational formula with an equality-conclusion and its previously generated premises, equates the terms in the equality-conclusion, i.e. one side is substituted by the other side. Chase procedures are designed to apply such rules, starting with the premises of the constraint to be decided, until no further change can be produced. If the procedure terminates, the constraint is implied iff its conclusion is among the finally produced statements.

In general, the implication problem for semantic constraints is fairly well understood in the relational case:

- As long as the constraints are "full", i.e., basically, in their implicational formulae no existentially quantified variable occurs positively, we have decidability.
- Otherwise, for "embedded" constraints, we have undecidability due to positively occuring existentially quantified variables. Such variables can cause the generation of an unlimited number of terms in executing proof procedures that do not terminate in this case.

Actually, even for the restricted case of embedded multivalued dependencies the implication problem has been proved to be undecidable [Her95]. The sophisticated proof employs a reduction of the word problem for finite semigroups, known to be undecidable, to the implication problem for embedded multivalued dependencies.

As already mentioned above, null values are important for the theory of schema decompositions [CM87, Heg94] and also for so-called "representative instances" of fragmented database schemas [Hon82, Sag83, GMV86]. Accordingly, the meaning of semantic constraints in the presence of relations with null values and the corresponding variant of the implication problem have been studied [Lie79, Vos79, Gra84, AM86, Tha91, LL94].

Theorem 6. *For the class of relation schemes $(\ , U, F)$, where F is a set of functional dependencies, the problem "$(\ , U, F)$ is in third normal form" is NP-complete.*

The deep reason for the negative result is that, in this situation, the problem "attribute A appears in a key of relation scheme $(\ , U, F)$" is already NP-complete; this result in turn is related to the fact that a relation scheme $(\ , U, F)$ can possibly have exponentially many keys [LO78, JF82, MR83, Kat92, VS93a, DKMST95].

Theorem 7. *For the class of relation schemes $(\ , U, F)$, where F is a set of functional dependencies, the problem "$(\ , U, F)$ is in Boyce/Codd normal form" is decidable in polynomial time.*

Indeed, a decision procedure can be based on the following equivalence [Osb78]: Boyce/Codd normal form iff for all $X \to Y \in F$ with $Y \not\subseteq X$: $X \to U \in F^+$. It should be noted, however, that also for Boyce/Codd normal form some important decision problems are of high computational complexity. In particular, deciding Boyce/Codd normal form for a *projection* of a scheme is coNP–complete [BB79]. This result on intractability as well as Theorem 6 contrast to the fact that the corresponding decision problems for relations, rather than schemes, are decidable in polynomial time (in the number of attributes and tuples of the relation under consideration) [DLM92].

Theorem 8. *For the class of relation schemes $(\ , U, F)$, where F is a set of functional dependencies, the problem "$(\ , U, F)$ is in unique key normal form" is decidable in polynomial time.*

Again, a decision procedure can be based on an equivalence statement [BDLM91]: unique key normal form iff $\{A \mid A \in U \text{ and } U \backslash A \to A \notin F^+ \} \to U \in F^+$.

5.6 Achieving Normal Forms and View Support Simultaneously

So far, the Separation of Aspects and the Inferential Completeness heuristics have been treated separately, although, as we have seen in Section 5.5, both heuristics lead to related implication problems. According to Task 2, we have to explore the relationship between their formalizations in more detail, in particular, whether their formal versions are compatible. As far as we can actually achieve the desirable syntactic properties simultaneously, according to Task 3, we have to design algorithms to obtain them.

The following two theorems are the most well-known examples of results on compatibility.

Theorem 9. *For every (universal relation) scheme $ES = (\ , U, F)$, where F is a set of functional dependencies, there exists a database schema DS with relation schemes $(\ , X_1, F_1), \ldots, (\ , X_n, F_n)$ for extensional enumerations such that:*
i) Schema DS supports *ES* by the natural join.
ii) Each scheme $(\ , X_i, F_i)$ of DS is in Boyce/Codd normal form.

Theorem 10. *For every (universal relation) scheme $ES = (\ , U, F)$, where F is a set of functional dependencies, there exists a database schema DS with relation schemes $(\ , X_1, F_1), \ldots, (\ , X_n, F_n)$ for extensional enumerations such that:*

i) Schema DS faithfully supports ES by the natural join.
ii) Each scheme $(, X_i, F_i)$ *of* DS *is in third normal form.*

The proofs of these and related theorems are constructive, yielding outlines of design methods of decomposition and synthesis, respectively [Cod72, Fag77, Fag81, Ber76, BB79, BDB79, KM80, LTK81, BK86, SR88, BM87, TLJ90, YÖ92a, YÖ92b]. Such methods will be discussed in a more general framework in Section 5.11.

5.7 Normal Forms Ensure Low Storage and Update Costs

We have introduced normal forms as desirable syntactic properties, formalizing the Separation of Aspects heuristics. According to Task 4, we now justify these normal forms in terms of cost, thus providing formal counterparts to informal motivations of the Separation of Aspects heuristic to avoid so-called "update anomalies".

The benefits of all purely decompositional normal forms in terms of *storage costs* are summarized as follows:

Theorem 11. *A relation scheme* $(, U, SC)$ *is in decompositional normal form (i.e. BCNF, 4 NF, 5 NF), relative to the class of semantic constraints considered in SC (i.e. functional dependencies, multivalued dependencies, join dependencies)*
iff for each decomposed database schema DS that supports $(, U, SC)$ *by the natural join, for each instance of* $(, U, SC)$*:*
size (instance of $(, U, SC)$*)* $\leq$ *size (decomposed instance).*

Here size means the number of occurences of constant symbols in the instances. This "folklore theorem" is closely related to a theorem of [VS93b] that characterizes normal forms in terms of data redundancy. The intuitive reasoning of the proof is the following. Assume that $(, U, SC)$ is already in decompositional normal form. Then any further decomposition would result in duplicating the key components of tuples (and thus would increase the size) without getting any compensating size benefit. If on the other hand $(, U, SC)$ is not in decompositional normal form, then the forbidden substructure can cause a redundant representation of facts, and the size benefit of removing this redundancy by decomposition can exceed the disadvantage of duplicating tuple components, necessary for support by the natural join.

Decompositional normal forms are also helpful to ensure low *update costs* [BG80, Vos88, Bis89, Cha89, HC91, BD93]. As an example, we present a theorem that takes care of functional and inclusion dependencies [BD93]. The theorem characterizes those database schemas that allow maintenance of all semantic constraints by simply checking whether a newly inserted tuple does not violate a key condition.

Theorem 12. *A database scheme DS with relation schemes $(R_1, X_1, F_1), \ldots, (R_n, X_n, F_n)$ with functional dependencies as local semantic constraints and inclusion dependencies I as global semantic constraints allows $X \subset X_i$, for some i, as* update object*, i.e.*

i) $X \to X_i \in (I \cup \bigcup_{i=1,\ldots,n} F_i)^+$ *and*

ii) *for each instance $(r_1, \ldots, r_i \ldots, r_n)$ of DS, for each tuple μ with $\mu\lceil X \notin \pi_X(r_i)$:*
$(r_1, \ldots, r_i \cup \{\mu\}, \ldots, r_n)$ *is instance of DS*

iff the following properties hold:

iii) R_i *is "not referencing" by inclusion dependencies of I.*

iv) R_i *is in unique key normal form.*

v) R_i *is in Boyce/Codd normal form.*

The proof is based on a careful analysis and equivalent reformulations of the fundamental notions, as well as, on separation conditions that restrict the interaction of functional dependencies and inclusion dependencies (which may be very complex in general, according to Theorem 5).

5.8 Unique Essences: Unique Flavor Formalized as Worthwhile "Semantic" Requirement

The fourth design heuristic, Unique Flavor, can be formalized in the framework of designing a so-called universal relation interface for a database schema [MUV84, KKFVU84, Var88b, BB83, BBSK86, BV88, Lev92]. Such an interface should translate queries, which are expressed in terms of attributes only (omitting the information about relation schemes), into join paths within the hypergraph structure of the schema. If there are several candidate join paths, then, according to the Unique Flavor heuristic, all these candidates should provide essentially the same query answer [BBSK86].

Given a database schema DS, a formalized version of this requirement is defined as follows:
$U := \bigcup_{i=1,\ldots,n} X_i$ and $H := \{X_1, \ldots, X_n\}$ describes the *hypergraph* of DS.
$jp : \wp U \to \wp\wp H$, $jp(Y) := \{E \mid E \subset H,\ E$ *connected*, $Y \subset \bigcup E,\ E$ *minimal*$\}$ defines the translation from a set of attributes into *join paths*.
$essence(E, Y) := \{X_i \cap (Y \cup \bigcup_{X_j \in E, X_j \neq X_i} X_j) \mid X_i \in E\}$ is the *essence* of a join path E over a set of attributes Y.

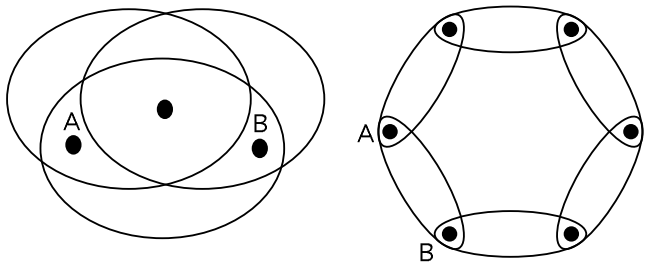
- Finally, Unique Flavor motivates that essences should be unique:
  For all $Y \subset U$, for all $E, F \in jp(Y) : essence(E, Y) = essence(F, Y)$.

Of course, here the property of unique essences is already defined in purely syntactic terms although it is "semantically" motivated.

5.9 Acyclicity: Unique Flavor Formalized as Desirable Syntactic Property

The fourth design heuristic, Unique Flavor, can also be rephrased by considering the hypergraph *structure* of a database schema, as defined by the formats, and by requiring that the hypergraph is to some degree acyclic [Fag83, BFMY83, BBSK86]. The two most important degrees are listed below by their names and their *forbidden substructures*:

– γ-*acyclic*:



– α-*acyclic*: a nontrivial hypergraph as produced by the GYO-reduction applied to the schema [Gra79, YÖ79].

As a contribution to Task 2, it turns out that γ-acyclicity syntactically characterizes the "semantic" property of Section 5.8 [BBSK86]:

Theorem 13. *DS is γ-acyclic iff DS has unique essences.*

We can easily construct join paths over some set of attributes Y with different essences from the cyclic substructures forbidden for γ-acyclic database schemas: in each case, $Y := \{A, B\}$ is contained in a single relation scheme, which constitutes a trivial covering join path, and Y is covered by an essentially different join path containing two or more relation schemes (namely those connecting A with B running the "long way"). The converse claim of the theorem is proved by a tedious and subtle examination of the so-called *intersection hypergraph* of DS, which is generated from the hypergraph by adding all nonempty intersections of its hyperedges.

As a contribution to Task 3, the desirable syntactic property of acyclicity can be efficiently decided [YÖ79, Gra79, Fag83, TY84, DM86]:

Theorem 14. *The problems "DS is γ-acyclic" and "DS is α-acyclic" are decidable in polynomial time.*

For each degree of acyclicity a recursive decision procedure can be based on a "pruning" predicate on hyperedges. At each stage the procedure can delete a hyperedge that satisfies the predicate. The initial schema is acyclic iff the procedure succeeds in reducing the schema to nothing. The pruning predicate for

γ-acyclicity can be paraphrased by "either there exists another hyperedge that has the same set of intersections with the remaining hyperedges or all nonempty intersections with the remaining hyperedges are identical". The pruning predicate for α-acyclicity can be paraphrased as "there exists a remaining hyperedge that contains all the intersections with the remaining hyperedges". These decision procedures are elaborated variants of the well-known acyclicity test for ordinary graphs: the test recursively deletes a leaf and its corresponding edge and recognizes acyclic graphs, i.e. trees, as those graphs that can be reduced to nothing.

5.10 Acyclicity Ensures Low Query Costs

According to Task 4, the impact of acyclicity as desirable syntactic property on costs should be examined. As suggested by the corresponding worthwhile semantic requirement, the evaluation of view queries that are join paths should be efficient. Indeed, we have the following assertions which have subtle and tedious proofs [Fag83, BFMY83]:

Theorem 15. *DS is γ-acyclic*
iff all *noncartesian join trees are* monotone
(i.e. for pairwise consistent relations $r_1, \ldots r_n$ all partial results are consistent)
iff projections $\pi_Y(u)$ with $u := \bowtie_{i=1,\ldots,n} r_i,\ r_i := \pi_{X_i}(u)$,
can be computed by determining a covering join path and evaluating it (i.e. $\pi_Y(u) = \pi_Y(\bowtie_{X_i \in E} r_i)$ for some $E \in jp(Y)$).

Theorem 16. *DS is α-acyclic*
iff there exists a monotone join tree
(which, essentially, is determined by the GYO-reduction).

5.11 Design Algorithms Remove Forbidden Substructures

Fig.5 summarizes the presented achievements with respect to Tasks 1, 2 and 4. Finally, according to Task 3, a lot of design methods for *achieving* desirable syntactic properties have been proposed. Apparently, any concrete refinement of such a method will, necessarily, result into a highly interactive design procedure. The general skeleton of such procedures and the division of labour between the insightful administrator and the automatic algorithm [Bis95, BC86, BC89, Bis95b] can be outlined as shown in Fig.6.

formalize heuristics

separation

Task 1. syntactic properties
no forbidden substructure:

- 3 NF
- BCNF
- 4 NF
- 5 NF
- referential NF
- unique key NF

unique flavour

Task 1. "semantic" requirements

- join paths are essentially unique

Task 1. syntactic properties
no forbidden substructure:

- γ-acyclic
- α-acyclic

inference

Task 1. semantic requirements

- (faithful) view instance support
- view query support

Task 2. syntactic characterization
for FDs, join support:

- $\bowtie [X_1, \ldots, X_n] \in SC^+$
- $(\bigcup_i F_i)^+ \supset F$

Task 2. relationships between syntactic properties and semantic requirements

- BCNF and join support are compatible (decomposition)
- 3 NF and faithful join support are compatible (synthesis)
- γ-acyclic iff join paths are essentially unique

optimize at design time

Task 4. syntactic properties ensure low costs

storage

- normal forms

query

- γ-acyclic:
  - join trees are monotone
  - projection by covering joins
- α-acyclic: existence of monotone join trees

update

- $\begin{cases} \text{not referencing} \\ \text{one key} \\ \text{BCNF} \end{cases}$

Fig.5. A short summary of presented achievements with respect to Tasks 1, 2 and 4.

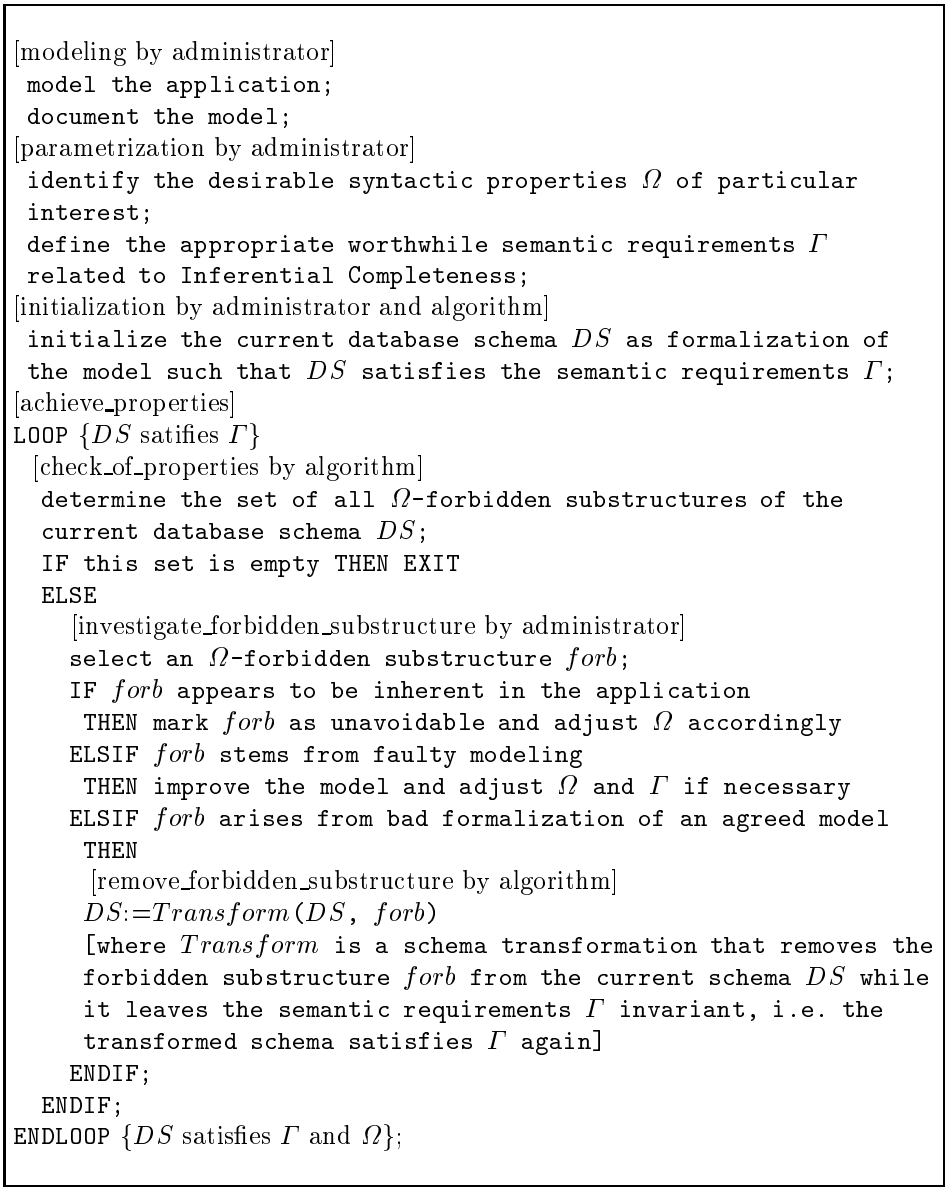


Fig.6. Outline of interactive design procedures.
[ ] embraces comments indicating a module together with its active entity or semantics, respectively;
{ } embraces state conditions, i.e. the invariant and the post condition of the main loop.

We shortly discuss some examples that fit the skeleton of Fig.6 by indicating Ω, Γ and $Transform$:

- Classical *decomposition*, based on Theorem 9 and related statements, identifies Ω as Boyce/Codd or higher normal form and selects Γ as view support of an initial universal relation scheme by the natural join. $Transform$ splits a current relation scheme with attribute set X_i into fragments that are determined by the components of the forbidden join dependency; $Transform$ is Γ-invariant by the semantics of join dependencies (and because any functional dependency implies a corresponding multivalued dependency).
- Classical *synthesis*, based on Theorem 10 and related statements, has two phases. The first phase achieves faithfulness, and the second phase adds view support of an initial universal relation scheme by the natural join. In the first phase Ω is identified as third normal form, and Γ is selected as faithfulness (syntactically characterized as the preservation of the given set of functional dependencies). $Transform$ removes all kinds of redundancy (which can lead to Ω-forbidden substructures) from the functional dependencies. An early version of synthesis [Ber76] achieves this goal by computing a minimal cover of the functional dependencies, leaving Γ invariant. A later version [BM87] achieves this goal by removing so-called "abnormal nonprime" attributes from relation schemes following some sophisticated strategy, whereby a somewhat stronger invariant Γ (namely faithfulness and "object-faithfulness" and "strong normativity") is preserved. In the second phase view support of an initial universal relation scheme by the natural join is guaranteed by ensuring the existence of a relation scheme that contains a global key [BDB79]. If necessary, such a key component is added (in this case as a previously "missing substructure").
- In *view integration*, as formalized in [BC86], Ω is identified as absence of so-called "integration constraints", which indicate a redundant overlap of views and, thus, are interpreted as forbidden substructures in the wanted integrated schema, and Γ is selected as view support of all given view database schemas. $Transform$ removes an integration constraint by appropriately merging the relation schemes involved, leaving Γ invariant.

Examining the general skeleton of design procedures, we can further comment on the tasks of design theory. The parametrization step is based on Task 1 and Task 2. The check_of_properties step and the remove_forbidden_substructure step are based on Task 3. Finally, after getting the final output schema, the administrator can evaluate the quality of the design based on results of Task 4, and, if necessary, he can iteratively process the design procedure using a different parametrization.

The skeleton also shows the impact of the administrator's interaction: he is responsible for modeling, parametrization, selection and investigation of forbidden substructure. While modeling and investigation of forbidden substructures is principally outside the scope of automatic algorithms, parametrization and selection of forbidden substructures could possibly be better supported by algorithms as known today.

6 Final Remarks

Since the first pioneering work of E.F. Codd [Cod70, 72] and W.W. Armstrong [Arm74], a substantial body of results on database schema design theory has been published. This paper summarizes only a small part of the highly detailed work. It emphasizes the tasks of design theory in producing and using interactive design tools. In a companion paper [Bis95a] three main topics for further enhancement of the design theory are outlined: all current achievements still have to be carried out within one unifying framework; the current achievements have to be embedded in the full design process and to be extended to deal more deeply with advanced database features like incomplete information, recursive query languages, complex objects or object identifiers; all achievements have to be reconsidered from the viewpoint of distributed computing, abandoning the classical centralized approach to databases.

Acknowledgement: I would like to thank Ralf Menzel and Torsten Polle for valuable discussions. I am also grateful to an anonymous reviewer for helpful remarks and hints.

References

[AABM82] P. Atzeni, G. Ausiello, C. Batini, and M. Moscarini. Inclusion and equivalence between relational database schemata. *Theoretical Computer Science*, 19:267–285, 1982.

[ADA93] P. Atzeni and V. De Antonellis. *Relational Database Theory.* Benjamin/Cummings, Redwood City, CA, 1993.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, MA, 1995.

[AM86] P. Atzeni and N. M. Morfuni. Functional dependencies and constraints on null values in database relations. *Information and Control*, 70(1):1–31, 1986.

[Arm74] W. W. Armstrong. Dependency structures of data base relationships. In J. L. Rosenfeld, editor, *Proceedings of IFIP Congress 1974*, pages 580–583. North-Holland, Amsterdam, 1974.

[BB79] C. Beeri and P.A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems*, 4(1):30–59, 1979.

[BB83] J. Biskup and H. H. Brüggemann. Universal relation views: A pragmatic approach. In *Proceedings of the 9th International Conference on Very Large Data Bases*, pages 172–185, 1983.

[BBG78] C. Beeri, P. A. Bernstein, and N. Goodman. A sophisticated introduction to database normalization theory. In *Proceedings of the 4th International Conference on Very Large Data Bases, Berlin*, pages 113–124, September 1978.

[BBG78] C. Beeri, P. A. Bernstein, and N. Goodman. A sophisticated introduction to database normalization theory. In *Proceedings of the 4th International Conference on Very Large Data Bases, Berlin*, pages 113–124, September 1978.

[BBSK86] J. Biskup, H. H. Brüggemann, L. Schnetgöke, and M. Kramer. One flavor assumption and γ-acyclicity for universal relation views. In *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 148–159, 1986.

[BC86] J. Biskup and B. Convent. A formal view integration method. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington*, pages 398–407, 1986.

[BC89] J. Biskup and B. Convent. Towards a schema design methodology for deductive databases. In J. Demetrovics and B. Thalheim, editors, *Proceedings of the Symposium on Mathematical Fundamentals of Database Systems (MFDBS '89)*, number 364 in Lecture Notes in Computer Science, pages 37–52. Springer, 1989.

[BC91] J. Biskup and B. Convent. Relational chase procedures interpreted as resolution with paramodulation. *Fundamenta Informaticae*, XV(2):123–138, 1991.

[BD93] J. Biskup and P. Dublish. Objects in relational database schemes with functional, inclusion and exclusion dependencies. *Informatique théorique et Applications / Theoretical Informatics and Applications*, 27(3):183–219, 1993.

[BDB79] J. Biskup, U. Dayal, and P. A. Bernstein. Synthesizing independent database schemas. In P. A. Bernstein, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '79), Boston*, pages 143–151, New York, NY, 1979. ACM.

[BDLM91] J. Biskup, J. Demetrovics, L. O. Libkin, and I. B. Muchnik. On relational database schemes having unique minimal key. *Journal of Information Processing and Cybernetics EIK*, 27(4):217–225, 1991.

[Ber76] P. A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Transactions on Database Systems*, 1(4):272–298, December 1976.

[BFMY83] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.

[BG80] P. A. Bernstein and N. Goodman. What does Boyce-Codd normal form do? In *Proceedings of the 6th International Conference on Very Large Data Bases*, pages 245–259, 1980.

[Bis80] J. Biskup. Inferences of multivalued dependencies in fixed and undetermined universes. *Theoretical Computer Science*, 10:93–105, 1980.

[Bis85] J. Biskup. Entwurf von Datenbankschemas durch schrittweises Umwandeln verbotener Teilstrukturen. In *Tagungsband GI-EMISA-Fachgespräch Entwurf von Informationssystemen — Methoden und Modelle, Tutzing*, pages 130–148, 1985.

[Bis89] J. Biskup. Boyce-Codd normal form and object normal forms. *Information Processing Letters*, 32(1):29–33, 1989.

[Bis94] J. Biskup. Impacts of creating, implementing and using formal languages. In K. Duncan and K. Krueger, editors, *Proceedings of the 13th World Comupter Congress 94*, volume 3, pages 402–407. Elsevier (North-Holland), Amsterdam etc., 1994.

[Bis95a] J. Biskup. Database schema design theory: achievements and challenges. In *Proceedings of the 6th International Conference on Information Systems and Management of Data (CISMOD '95)*, number 1006 in Lecture Notes in Computer Science, pages 14–44, Bombay, 1995. Springer, Berlin etc.

[Bis95b] J. Biskup. *Grundlagen von Informationssystemen.* Vieweg, Braunschweig-Wiesbaden, 1995.

[BK86] C. Beeri and M. Kifer. An integrated approach to logical design of relational database schemes. *ACM Transactions on Database Systems*, 11(2):134–158, 1986.

[BM87] J. Biskup and R. Meyer. Design of relational database schemes by deleting attributes in the canonical decomposition. *Journal of Computer and System Sciences*, 35(1):1–22, 1987.

[BR88] J. Biskup and U. Räsch. The equivalence problem for relational database schemes. In *Proceedings of the 1st Symposium on Mathematical Fundamentals of Database Systems*, number 305 in Lecture Notes in Computer Science, pages 42–70. Springer-Verlag, Berlin etc., 1988.

[BS81] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981.

[BV84a] C. Beeri and M. Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM Journal on Computing*, 13(1):76–98, 1984.

[BV84b] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, October 1984.

[BV88] V. Brosda and G. Vossen. Update and retrieval in a relational database through a universal schema interface. *ACM Transactions on Database Systems*, 13(1988):449–485, 1988.

[CFP84] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*, 28(1):29–59, 1984.

[Cha89] E. P. F. Chan. A design theory for solving the anomalies problem. *SIAM Journal on Computing*, 18(3):429–448, June 1989.

[Che76] P. P.-S. Chen. The entity-relationship-model — towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[CM87] E. P. F. Chan and A. O. Mendelzon. Independent and separable database schemes. *SIAM Journal on Computing*, 16(5):841–851, 1987.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[Cod72] E. F. Codd. Further normalization of the database relational model. In R. Rustin, editor, *Database Systems*, number 6 in Courant Institute Computer Science Symposia Series, pages 33–64. Prentice Hall, Englewood Cliffs, NJ, 1972.

[CV85] A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM Journal on Computing*, 14(3):671–677, 1985.

[DB82] U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems*, 8(3):381–416, 1982.

[Del78] C. Delobel. Normalization and hierarchical dependencies in the relational data model. *ACM Transactions on Database Systems*, 3:201–222, 1978.

[DF92] C. J. Date and R. Fagin. Simple conditions for guaranteeing higher normal forms in relational databases. *ACM Transactions on Database Systems*, 17:465–476, 1992.

[DKM+95] J. Demetrovics, G. O. H. Katona, D. Miklos, O. Seleznjew, and B. Thalheim. The average length of key and functional dependencies in (random)

databases. In G. Gottlob and M. Y. Vardi, editors, *Database Theory—ICDT '95*, pages 266–279. Springer-Verlag, Berlin etc., 1995.

[DLM92] J. Demetrovics, L. Libkin, and I.B. Muchnik. Functional dependencies in relational databases: a lattice point of view. *Discrete Applied Mathematics*, 40:155–185, 1992.

[DM86] A. D'Atri and M. Moscarini. Recognition algorithms and design methodologies for acyclic database. In P. C. Kanellakis and F. Preparata, editors, *Advances in Computing Research*, volume 3, pages 164–185. JAI Press, Inc., Greenwich, CT, 1986.

[DP84] P. DeBra and J. Paredaens. Horizontal decompositions for handling exceptions to functional dependencies. In H. Gallaire, J. Minker, and J. M. Nicolas, editors, *Advances in Database Theory*, volume 2. Plenum, New York - London, 1984.

[Fag77] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems*, 2(3):262–278, September 1977.

[Fag81] R. Fagin. A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems*, 6(3):387–415, 1981.

[Fag83] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, July 1983.

[FC85] A. L. Furtado and M. A. Casanova. Updating relational views. In W. Kim, D. S. Reiner, and D. S. Batory, editors, *Query Processing in Database Systems*. Springer-Verlag, Berlin, 1985.

[FV84] R. Fagin and M. Y. Vardi. The theory of data dependencies - an overview. In *Proceedings of the 11th International Colloquium on Automata, Languages and Programming*, number 172 in Lecture Notes in Computer Science, pages 1–22. Springer-Verlag, Berlin etc., 1984.

[GHLM93] J. Grant, J. Horty, J. Lobo, and J. Minker. View Updates in Stratified Disjunctive Databases. *Journal of Automated Reasoning*, 11:249–267, 1993.

[GMV86] M. H. Graham, A. O. Mendelzon, and M. Y. Vardi. Notions of dependency satisfaction. *Journal of the ACM*, 33(1):105–129, 1986.

[Gra79] M. H. Graham. On the universal relation. Systems research group report, University of Toronto, 1979.

[Gra84] G. Grahne. Dependency satisfaction in databases with incomplete information. In U. Dayal, editor, *Proceedings of the 10th International Conference on Very Large Data Bases*, pages 37–45, Singapore, 1984.

[HC91] H. J. Hernández and E. P. F. Chan. Constant-time-maintainable BCNF database schemes. *ACM Transactions on Database Systems*, 16(4):571–599, December 1991.

[Heg94] S. J. Hegner. Unique complements and decompositions of database schemata. *Journal of Computer and System Sciences*, 48:9–57, 1994.

[Her95] C. Herrmann. On the undecidability of implications between embedded multivalued dependencies. *Information and Computation*, 122:221–235, 1995.

[Hon82] P. Honeyman. Testing satisfaction of functional dependencies. *Journal of the ACM*, 29(3):668–677, 1982.

[Hul86] R. Hull. Relative information capacity of simple relational database schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986.

[JF82] J. H. Jou and P. C. Fischer. The complexity of recognizing 3NF relation schemes. *Information Processing Letters*, 14(4):187–190, 1982.

[Kat92] G. O. H. Katona. Combinatorial and algebraic results for database relations. In *Database Theory—ICDT '92*, number 646 in Lectures Notes in Computer Science, pages 1–20. Springer-Verlag, Berlin etc., 1992.

[KCV83] P. C. Kanellakis, S. S. Cosmadakis, and M. Y. Vardi. Unary inclusion dependencies have polynomial time inference problems. In *Proceedings of the 15th Symposium on Theory of Computing, Boston*, pages 246–277, 1983.

[Kel86] A. M. Keller. The role of semantics in translating view updates. *IEEE Computer*, 19(1):63–73, January 1986.

[Ken83] W. Kent. A simple guide to five normal forms in relational databases. *Communications of the ACM*, 26(2):120–125, 1983.

[KKF+84] H. F. Korth, G. M. Kuper, J. Feigenbaum, A. VanGeldern, and J. D. Ullman. A database system based on the universal relation assumption. *ACM Transactions on Database Systems*, 9(1984):331–347, 1984.

[KM80] P. Kandzia and M. Mangelmann. On covering Boyce-Codd normal forms. *Information Processing Letters*, 11:218–223, 1980.

[Lev92] M. Levene. *The Nested Universal Relation Database Model*. Lecture Notes in Computer Science 595. Springer, Berlin etc., 1992.

[Lie79] Y. E. Lien. Multivalued dependencies with nulls in relational databases. In *Proceedings of the 5th International Conference on Very Large Data Bases*, pages 61–66, 1979.

[LL94] M. Levene and G. Loizou. The nested universal relation model. *Journal of Computer and System Sciences*, 49:683–717, 1994.

[LO78] C. L. Lucchesi and S. L. Osborn. Candidate keys for relations. *Journal of Computer and System Sciences*, 17(2):270–279, 1978.

[LTK81] T.-W. Ling, F. W. Tompa, and T. Kameda. An improved third normal form for relational databases. *ACM Transactions on Database Systems*, 6(2):329–346, 1981.

[Mai83] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, MD, 1983.

[Men79] A. O. Mendelzon. On axiomatizing multivalued dependencies in relational databases. *Journal of the ACM*, 26(1):37–44, 1979.

[Mit83] J. C. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56(3):154–173, 1983.

[MR83] H. Mannila and K.-J. Räihä. On the relationship of minimum and optimum covers for a set of functional dependencies. *Acta Informatica*, 20:143–158, 1983.

[MR86] H. Mannila and K.-J. Räihä. Inclusion dependencies in database design. In *Proceedings of the Second International Conference on Data Engineering*, pages 713–718, Washington, DC, 1986. IEEE Computer Society Press.

[MR92] H. Mannila and K.-J. Räihä. *The Design of Relational Databases*. Addison-Wesley, Wokingham, England, 1992.

[MUV84] D. Maier, J. D. Ullman, and M. Y. Vardi. On the foundations of the universal relation model. *ACM Transactions on Database Systems*, 9(2):283–308, June 1984.

[Osb78] S. L. Osborn. *Normal Forms for Relational Data Bases*. PhD thesis, Department of Computer Science, University of Waterloo, 1978.

[PDGvG89] J. Paredaens, P. DeBra, M. Gyssens, and D. van Gucht. *The Structure of the Relational Database Model.* Number 17 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1989.

[Ris77] J. Rissanen. Independent components of relations. *ACM Transactions on Database Systems*, 2(4):317–325, 1977.

[Ris82] J. Rissanen. On equivalence of database schemes. In *Proceedings of the 1st ACM SIGACT–SIGMOD Symposium on Principles of Database Systems*, pages 23–26, 1982.

[Sag83] Y. Sagiv. A characterization of globally consistent databases and their correct access paths. *ACM Transactions on Database Systems*, 8(2):266–286, 1983.

[SR88] D. Seipel and D. Ruland. Designing gamma-acyclic database schemes using decomposition and augmentation techniques. In *Proc. 1st Symposium on Mathematical Fundamentals of Database Systems*, number 305 in Lecture Notes in Computer Science, pages 197–209. Springer-Verlag, Berlin etc., 1988.

[Tha91] B. Thalheim. *Dependencies in relational databases.* Teubner, Stuttgart - Leipzig, 1991.

[TLJ90] P. Thanisch, G. Loizou, and G. Jones. Succint database schemes. *International Journal of Computer Mathematics*, 33:55–69, 1990.

[TY84] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectivity reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13:566–579, 1984.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems (Volume I).* Computer Science Press, Rockville, MD, 1988.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems (Volume II: The New Technologies).* Computer Science Press, Rockville, MD, 1989.

[Var82] M. Y. Vardi. On decomposition of relational databases. In *Proc. 23rd Symposium on Foundations of Computer Science*, pages 176–185, 1982.

[Var84] M. Y. Vardi. The implication and finite implication problem for typed template dependencies. *Journal of Computer and System Sciences*, 28:3–28, 1984.

[Var88a] M. Y. Vardi. Fundamentals of dependency theory. In E. Börger, editor, *Trends in Theoretical Computer Science*, pages 171–224. Computer Science Press, Rockville, 1988.

[Var88b] M. Y. Vardi. The universal-relation data model for logical independence. *IEEE Software*, 5(1988):80–85, 1988.

[Vas79] Y. Vassiliou. Null values in database management, a denotational semantics approach. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 162–169, 1979.

[Vos88] G. Vossen. A new characterization of FD implication with an application to update anomalies. *Information Processing Letters*, 29(3):131–135, 1988.

[Vos91] G. Vossen. *Data Models, Database Languages and Database Management Systems.* Addison-Wesley, Wokingham, England, 1991.

[VS93a] M. W. Vincent and B. Srinivasan. A note on relation schemes which are in 3NF but not in BCNF. *Information Processing Letters*, 48:281–283, 1993.

[VS93b] M. W. Vincent and B. Srinivasan. Redundancy and the justification for fourth normal form in relational databases. *International Journal of Foundations of Computer Science*, 4:355–365, 1993.

[YÖ79] C. T. Yu and Z. M. Özsoyoğlu. An algorithm for tree-query membership of a distributed query. In *Proceedings of the 3rd IEEE COMPSAC, Chicago*, pages 306–312, 1979.

[YÖ92a] L.-Y. Yuan and Z. M. Özsoyoğlu. Design of desirable relational database schemes. *Journal of Computer and System Sciences*, 45:435–470, 1992.

[YÖ92b] L.-Y. Yuan and Z. M. Özsoyoğlu. Unifying functional and multivalued dependencies for relational database design. *Information Science*, 59:185–211, 1992.

[Zan76] C. Zaniolo. *Analysis and design of relational schemata for database systems.* PhD thesis, University of California Los Angeles, Computer Science Department, 1976. Technical Report UCLA-ENG-7669, July 1976.

This article was processed using the LaTeX macro package with LLNCS style