Eötvös Loránd University
Faculty of Informatics

A General Partition Data Model and a Contribution to the Theory of Functional Dependencies

Theses of PhD Dissertation

Author: András Molnár

Supervisor: Prof. Dr. András Benczúr
Head of the Department of Information Systems

Doctoral School of Informatics
Director: Prof. Dr. János Demetrovics

Doctoral Program: Information Systems
Program Director: Prof. Dr. András Benczúr

2007.

1 Preliminaries and Aims

Design of database structuring is based on modeling the real world. The concept of *data model* has two meanings: on one hand, it means a database paradigm, a language with abstract data types that is used for specifying database schemata. The traditional *relational model* is one example of it [Cod70, Ull89, AHV95]. A database management system provides a high-level abstraction layer for handling data, corresponding to its data model. Therefore, the user or application developer does not need to care about e.g. physical storage or query algorithms.

On the other hand, data model refers to a *conceptual model* of a particular application domain, which describes the real-world entities (objects), their properties and relationships. The *Entity-Relationship* model [Ull89] is probably the most popular graphical language for expressing a conceptual model. The type of a relationship is determined by the number of participating entities, optionality of participation and the possible number of entities of one class that can be related to an entity in another class. These cardinality constraints must be precisely acquired and declared during the modeling process in order to specify the relationship type, to avoid redundancy and anomalies by later refinement and normalization of the shema.

One of the basic types of such cardinality constraints is *functional dependency* [Cod70], which expresses in terms of a database table that the values of one or more specific attributes (columns) uniquely determine another attribute. In terms of binary relationships, it means an entity of one class can only be related to at most one entity of the other class via the relationship. If such constraint is not prescribed, it may be denoted by a *negated dependency (excluded functional constraint)*, serving for checking completeness and consistency of the specification. Given two entity classes four kinds of relationships may exist between them. They form three basic types (*1:1, m:n, 1:n* and its symmetric pair *n:1* of the same basic type).

Common graphical modeling languages are lacking of expressivity regarding to relationships with higther arity. They cannot express all possible types, do not deal with logical implications or consistency. Higher-arity relationships are often decomposed, normalized. The most suitable decomposition can be obtained only if all dependencies are found and the type of the relationship is declared based on them. For instance, [Cam02] shows all the 14 types of ternary relationships (some of them can be decomposed). But what about the higher-arity cases? [Tha87].

Although functional dependencies have a rich and well-founded theoretical background, their traditional notation sometimes leads to redundancy and difficulties. One of the aims of my research work is to make the management of constraints simpler, more surveyable. Closed constraint sets should be classified and enumerated. To achieve this, suitable representation and reasoning systems are sought (e.g. graphical representation).

Recently, *data warehousing and multidimensional databases* [HK01] have gained high significance both in theory and practice. The *multidimensional data model* classifies real-world events, observations (e.g. transactions, measurements) by different aspects in parallel. These classification aspects form *dimensions*, each usually having a hierachical structure of granularities of detail. A *data cube* contains aggregated metric or descriptive data for each combination of granularity level values of dimensions.

In the case of multidimensional databases it is usually assumed that dimension hierarchy is more or less homogenous and all data is available in the lowest level of the hierarchy, i.e. the cube with metric data is valid and known for the whole dataset in the finest granularity. Due to the nature of the application or the lack of information, this may be heterogenous, e.g. some dimensions or granularity levels are invalid or unknown for some parts. The ability to define a

proper schema describing the structure and the available data is important in such cases. It should allow smart utilization of existing (*extensional*) data for computing derivable (*intensional*) data. Moreover, it should determine the valid, computable navigation and aggregation operations and ensure consistent extension of data as soon as more information becomes known.

An important property from the point of aggregation is *summarizability*. It refers to the computability of an aggregation value of a set using the aggregation values of its disjoint components. To achieve summarizability, one needs to safely determine whether the given sets are really disjoint and the semantics of aggregation is meaningful.

A data cube can be treated as a classification of the elements of a real-world population into disjoint set systems, with aggregation data attached to the elementary set intersections. It is a multiway, multi-level *partitioning*. Spyratos [Spy87] introduced a special semantics for the traditional relational model, *partition semantics*. The other aim of my work is to improve this model, survey the possibilities of a general partition database model, in order to manage the naming structure of disjoint set systems and their aggregation data. The concept of partitions appears in fields other than data warehousing, e.g. *geographical fields* in GIS are often interpreted as partitions.

The two aims are strongly related to each other since the refinement structure of partitions can be described by sets of functional dependencies in general, which also model traditional dimension hierarchy as a special case.

2 Applied Methods

The starting points for the foundation of general partition database model were the logical-deductive partition model of Spyratos [Spy87] and the concept of data cube. The aim is an application-independent partition model; in particular, generalization of the data cube for heterogenous structures, missing information, and in a longer term, metadata handling.

In the traditional relational model, different values of an attribute naturally partition the tuples of a relational table. Partition semantics introduce an extra abstraction level in the data model, since relations contain data about sets of real-world entities (elementary intersections) and not the entities themselves. Concepts of the relational model must be re-interpreted with this semantics and the differences revealed. A concept referring to a partition relation must also be interpreted in terms of the partitioned elements in parallel. I have investigated the possibilites of organizing partition relations into databases and constructing a set of operations for them. To survey and motivate this, several basic cases have been formulated and analyzed.

I have systematically examined the traditional relational algebra operations how their conventional syntax can be interpreted, what external information determines their validity and what kind of new concepts need to be introduced for the use with partition semantics.

During the research of connections between functional dependencies and partitions I have used the concept of *conjunctive dependency* of Spyratos [Spy87] and the known but rarely used semilattice representation of sets of functional dependencies [DLM89]. All this motivates a deeper analysis of the constraint sets.

In the thesis I present a disadvantageous example in the traditional functional dependency formalism using the well-known *Armstrong axiomatization* [AHV95, Ull89] which allows derivation of a non-trivial result only if a pseudo-trivial constraint (a constraint which has an attribute appearing in its both sides) is derived during the implication process.

In order to enumerate and reach a more feasible management of functional constraint sets I have defined a simplified syntax of constraints by getting rid of some redundancies. More suitable

axiomatizations needed to be sought than the traditional Armstrong system and their soundness and completeness must have proven in terms of the simplified syntax, using such properties of the Armstrong system in terms of the traditional syntax.

The resulting methods have made possible to generate each possible constraint set by a program for a given number of attributes.

To make constraint sets more surveyable, possibilities of visual presentation and reasoning have been considered. Spreadsheet and graphical representations of constraint sets were introduced and tried to be generalized for higher number of attributes.

3 Theses of the Dissertation

3.1 General Partition Data Model

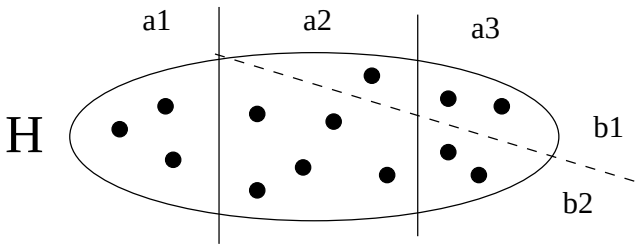
1. In Chapter 4 of my dissertation I reformulate the logical-deductive *partition data model* of Spyratos [Spy87] in an algebraic fashioned syntax and extend it with the concepts of *base sets* and *aggregation attributes* [1]. This special semantics makes the well-known generic relational data model become a language for set description management, capable for coding real-world elements into finite disjoint set systems, management of set names (codes) and correct handling, summarizing of statistical (aggregation) data, based on the principle of disjointness. Only the simplest summarizable (cumulative) type of aggregation is considered yet.

   Elementary symbols in a relation are set names, columns correspond to partitions, tuples to elementary set intersections. A partition database is built of such intersections and the model aims at a proper denotation of disjointness among sets. Figure 1 shows an example of a partition relation with two partition attributes over base set H. Partition A is made of sets a_1, a_2, a_3 and partition B is of b_1, b_2. Nonempty intersections of A, B pairs are stored in relation R, extended with a cumulative attribute N, currently referring to the cardinality of sets.

   Partition semantics contain the mapping of set names to real sets. It is not contained by a – strictly taken – partition database. Valid operations are those that can be performed on this level, without taking into account population-level (raw) data. These are treated as external knowledge.

   As a result, we get a relational description model that utilizes the tabular representation of data in set naming and disjointness. Base sets of different relations may differ and a partition schema may contain cumulative attributes. *Compatibility of semantics* is introduced for declaration of comparability among relations, since the relationships of attributes and symbols do not automatically follow from the syntax of relations.

2. Chapter 4 also demonstrates that a naive usage of relational algebra with partition semantics can easily be misleading and the operations do not always give meaningful results. The primary reason is that the operations on the relations are to be interpreted in terms of the set systems they represent. The *selection-projection problem* makes clear that partition relations themselves do not always contain enough information to perform some operations. External knowledge is required in such cases to determine the relationship of partitions represented by the operand relations. The *join problem* shows there are cases when the full partition lattice cannot be constructed. Instead, incomparable partitions may be present at the same time.



R:

A	B	N
a1	b2	3
a2	b1	1
a2	b2	5
a3	b1	2
a3	b2	2

Figure 1: Example of a partition relation

In my thesis I construct a simple partition algebra that is closed over partition relations. Cumulative attributes are handled automatically in the background. If the partition semantics of the operands are known, the partition semantics of the result can be determined. Base sets of relations and their relationships must be recorded, managed and considered while constructing algebra expressions. I introduce the formalism of *base set constraints* as an explicit means of this. They constrain the partition semantics of relations, i.e. the mapping of set names to real sets. Base set constraints can also be viewed as strict versions of compatibility of semantics. Since the particular mapping is not stored on this level, a base set constraint can be treated as metadata. By the way, it may have a syntactic aspect for the relations which is a necessary and sufficient condition for the existence of proper partition semantics obeying the base set constraint. Syntactical aspects are given for each type of base set constraints [1].

3. Also in Chapter 4, I briefly sketch the ability of functional dependencies to describe refinement structures of partitions and to determine aggregation and navigation possibilities. This is discussed in more detail in Chapter 5 and [1].

   Partition combinations (multiway partitions) denoting different elementary intersections correspond exactly to the closed attribute sets of functional dependencies among partition attributes. The semilattice of closed sets corresponds to the partition lattice and can be represented by a graph. Its paths correspond to the possible aggregation paths. I have introduced this notation into the partition model as the *structural graph* of partitions.

   Addition or removal of a partition attribute in a schema usually means real refinement or aggregation. In fact, if the closure of the resulting attribute set equals the original schema, no real navigation is performed along the partition lattice and the operation makes only formal difference in set names. The actual situation can be determined using the the structural graph.

   In a special case, the structural graph corresponds to the usual cuboid lattice. The left side of Figure 2 (omit the dashed line) shows such an example for partitions S, Z, D without any dependency. Vertical decomposition of the graph gives the dimension hierarchies, if there are any. The notation allows expressing more complex refinement structures.

   In addition to all above, I have extended the notation of the partition structural graph for the case if incomparable partitions are given for the same base set. Such a *partial graph* can be seen on Figure 2: if the partition according to SZD is know, we get the full graph as on the left side. If, however, only the partitions SZ, ZD are known then the part under the dashed line cannot be computed and the graph becomes cut as on the right side.

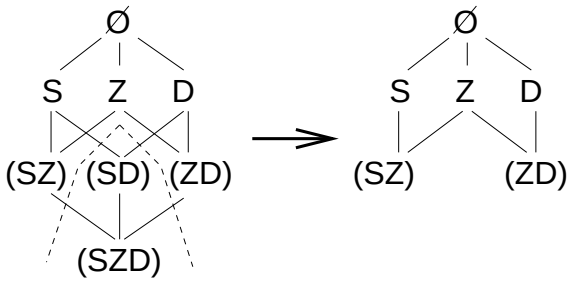


Figure 2: Full and partial structural graphs of partitions

I have extended the graph notation further for the case if some partitions are applicable only to subsets of the full base set. They can be represented by *subsetting edges*. It induces the introduction of *equivalence edges* as well, which can be used for declaring local dependencies, although the impact and utilization of local dependencies is not fully investigated yet.

4. At the end of Chapter 5 I sketch a semi-automatic decomposition method for creating a partition relational database schema given a structural graph of partitions. Interactivity may be important by some steps since it allows even the choice of redundant (denormalized) decomposition in case it is needed for effectiveness.

   As a result of the process, partition algebra expressions are attached to nodes of the structural graph. Nodes correspond to the different possible partitions and the expressions are view definitions on how to obtain them. User navigation (similarly to the traditional roll-up, drill-down or slice [HK01]) operations can be interpreted in terms of the structural graph as base set specification (selection) and attribute set specification (projection). These operations can be transformed automatically into partition algebra using the attached expressions of the nodes. User navigation is independent of the relational representation since the attached expressions can be freely changed.

5. In Chapter 6 I sketch the proposed formal framework of a general, application-independent partition data model, based on Chapters 3, 4 and 5. It has four levels as follows: 1^{st}*: structural (conceptual),* 2^{nd}*: logical (administration),* 3^{rd}*: instance (data),* 4^{th}*: interpretation.* Levels can be interpreted more or less similarly to the schema and interpretation level in the traditional relational model. These correspond here to levels 2 and 3, respectively. Level 1 desrcibes the (theoretically) valid partition structure as a structural graph. Known information is formulated as a partition relational schema on Level 2. Algebra expressions are attached to the structural graph on this level. An arbitrary (and possibly redundant) decomposition can be chosen on this level, not influencing the user operations of Level 1. Actual data as partition relations are a matter of Level 3. Partition semantics (resolving the set names used in the database symbols) belong to Level 4. The following structural elements of this four-level formalism can be theoretically changed, extended (*plug-ins*): inherent (algebraic) structure of set names, set definition and partition definition expression syntax, aggregation definition.

6. In order to survey the facilities of the partition model, I formulate several basic cases in Chapter 6 of the thesis, which demonstrate the descriptive power of the model (e.g. selection of an

arbitrary part of a data cube and refine its partitioning by a new classification). By analyzing the basic cases I have specified *basic partition language operations* (elementary constructors and queries) in a procedural fashion (see Appendix A). Their syntax is given as signatures, their semantics only informally yet. They are grouped according to the level(s) which they affect. First-level operations are based on attributes and base sets and are independent of the particular relational representation. The basic cases can be realized using the defined operations.

This abstract language and set of operations can be treated as the basis of general partition database model. Definition of simple multidimensional schemata, heterogenous partition structures and missing information is possible. Chapter 6 also presents future refinement and improvement issues, such as the complete formal foundation, effects of levels to each other, local dependencies, data manipulation, shared partition databases, exact specification of plug-ins, smart logical deduction for the second level, completeness of operations, extended aggregation facilities and the applicability of the model to real examples with further needs. The partition model in an improved form may be a paradigm for the formal description of the theory of data warehouses and aggregations in a longer term.

3.2 A Contribution to the Theory of Functional Constraints

1. Entity-Relationship [Ull89] and other common graphical modeling languages have restricted capabilities to express cardinalities in higher-arity relationships. For precise specification, schema refinement and normalization, one needs to represent further database constraints. The aim is to treat sets of constraints in a surveyable form instead of representing constraints one-by-one.

   By derivation in the traditional formalism of functional dependencies – especially when using the Armstrong axiomatization [AV85] and its extension to negated constraints [Tha00] – one often faces redundancies and difficulties. To make it simpler, I excluded the trivial and non-singleton constraints from the syntax. The naive restriction of the Armstrong axiomatization is, however, for the simplified syntax not complete any more.

   In Chapter 7 of my thesis I present the following axiomatization for the simplified syntax and prove its soundness and completeness (*ST/PQRST axiomatizations*, see also [2]). Y is the only attribute set variable and A, B, C are different attributes not occuring in Y.

   (S) $\frac{Y \to B}{YC \to B}$ (T) $\frac{Y \to A, YA \to B}{Y \to B}$

   (P) $\frac{YC \twoheadrightarrow B}{Y \twoheadrightarrow B}$ (Q) $\frac{Y \to A, Y \twoheadrightarrow B}{YA \twoheadrightarrow B}$

   (R) $\frac{YA \to B, Y \twoheadrightarrow B}{Y \twoheadrightarrow A}$ ($\square$) $\neg(Y \to B, Y \twoheadrightarrow B)$

   Soundness and completeness is verified using such properties of the Armstrong system, via a new intermediate axiomatization, the *U system* (for negated constraints via the systems *UE* and *NST*). For instance, to prove completeness of ST I take an arbitrary derivation that uses rules of the Armstrong system and construct an equivalent derivation in the simplified syntax using rule U. Finally the equivalence of U and ST is verified.

   Among several advantageous properties of the ruleset the most important is the order of rule application: a specific order of rules (its regular expression is $(S)^*; (T)^*; (R)^*; ((P) \| (Q))^*$)

forms a complete derivation method (ST/STRPQ algorithm). I have verified the completeness of this ordering by construction in Chapter 7 (Theorem 3). Based on the rules, I present a method of adding a single constraint to a closed constraint set (Chapter 10, see also [4]).

For classification reasons, I propose the concept of *dimension of a constraint* as the size of its left side and the *dimension of an attribute* as the size of a minimal attribute set that functionally determines the attribute (or if not determined at all, it is ∞ by default).

2. Graphical and spreadsheet representations of functional constraint sets for small attribute sets are introduced in Chapters 8 and 9 (see also [2, 3, 4, 5, 6]). Each possible dependency corresponds to a cell in the table and a node of a geometrical shape.

   Figure 3 shows some examples of the graphical (*triangular*) representation. On the left side, constraint $A \rightarrow B$ is depicted by a black circle with its implied constraint $AC \rightarrow B$ as an empty circle. The middle triangular shows the constraint set generated by $\{A \rightarrow B,\ B \rightarrow C\}$. The right side shows the negated dependency $AC \not\rightarrow B$ and its implications $A \not\rightarrow B$ and $C \not\rightarrow B$.

   This representation can be theoretically generalized for higher number of attributes. Triangular representation of n attributes exists in the $n-1$ dimensional space. For $n = 4$ we get a tetrahedron, which can be transformed into a planar form. The more attributes we have, the more difficult such a transformation is. The higher order representations have theoretical significance and may serve as basis of a data structure for effectively storing constraint sets.

   I present how *graphical reasoning* is possible in the triangular representation using graphical patterns of the implication rules. Reasoning is also possible in terms of the spreadsheet form. The semilattice representation of closed sets [DLM89] is an alternative graphical representation (it is used as the structural graph in the partition model). I give conversion methods for this representation in Chapter 10.

   These results help in further survey and classification of constraint sets and suggest the future implementation of a design tool software. Other types of database constraints (e.g. multivalued dependencies) might also be handled with a similar approach.

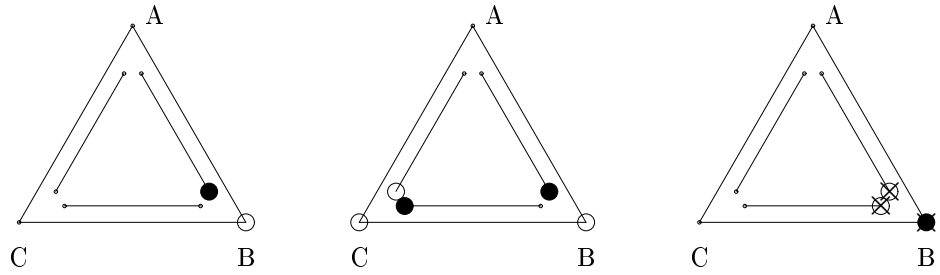


Figure 3: Examples of the triangular representation of constraint sets

3. In order to enumerate the closed sets of functional dependencies, I have written a PROLOG program, which is attached to the thesis as Insert (see also [2]). The procedure is based on the ST ruleset and systematically generates the possible closed constraint sets by attribute dimensions. The core is the generation of constraints with minimal left sides and using rule (S) only. If rule (T) is applicable, processing of the set is immediately aborted because such sets will be generated starting from minimal left sides using rule (S) only. I have optimized

n	$\lvert\mathcal{SD}_n\rvert$	$\lvert\mathcal{SD}_n/\tau\rvert$	$\lvert\mathcal{SD}^0_n\rvert$	$\lvert\mathcal{SD}^0_n/\tau\rvert$
1	1	*1*	2	2
2	4	*3*	7	5
3	45	*14*	61	19
4	2 271	*165*	2 480	184
5	1 373 701	*14 480*	1 385 552	14 664

Table 1: Number of closed sets of functional dependencies for n attributes. The column printed in italics shows the different cases with no constant attribute definitions (they correspond to n-ary relationship types)

the program and managed to run it for 3, 4 and 5 attributes. Results are shown on Table 1 [3, 4].

The first column of the table shows the number of closed functional dependency sets for n attributes (with no dependencies declaring constant attributes). If roles of attributes are not fixed then we get the equivalent classes (cases) up to permutation. They correspond to the different types of n-ary relationships. The program computes the number of such types as well. The second column of the table shows the values.

The third and fourth columns of the table show these numbers if constant attribute definitions (0 dimensional constraints) are allowed.

The exact number of theoretical relationship types and functional dependency sets is still unknown for arbitrary n, only some estimations exist [BDK$^+$91, DLM89]. Meanwhile, [HN05] is published, which determines one of the four numbers of my table for $n = 6$.

Publications of the Author

[1] A. Benczúr and A. Molnár. An extended partition model for generalized multidimensional data. Technical Report 2/2007, eScience Regional Knowledge Center, Eötvös L. University, Budapest, 2007.

[2] J. Demetrovics, A. Molnár, and B. Thalheim. Graphical and spreadsheet reasoning for sets of functional dependencies. Technical Report 0404, Kiel University, Computer Science Institute, http://www.informatik.uni-kiel.de/reports/2004/0404.html, 2004.

[3] J. Demetrovics, A. Molnár, and B. Thalheim. Graphical reasoning for sets of functional dependencies. In *Proceedings of ER 2004, Lecture Notes in Computer Science 3288*, pages 166–179, Shanghai, China, 2004. Springer Verlag.

[4] J. Demetrovics, A. Molnár, and B. Thalheim. Relationship design using spreadsheet reasoning for sets of functional dependencies. In *Proceedings of ADBIS 2006, Lecture Notes in Computer Science 4152*, pages 108–123, Thessaloniki, Greece, 2006. Springer Verlag.

[5] J. Demetrovics, A. Molnár, and B. Thalheim. Relációs adatbázisok funkcionális függõségeinek grafikus axiomatizációja. *Alkalmazott matematikai lapok*, 2007, Budapest (accepted for publication).

[6] J. Demetrovics, A. Molnár, and B. Thalheim. Graphs representing sets of functional dependencies. *Annales Univ. Sci. Budapest, Sectio Computatorica*, 28, 2008, (accepted for publication).

Further Selected Bibliography

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

[AV85] S. Abiteboul and V. Vianu. Transactions and integrity constraints. In *Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems - PODS'85*, pages 193–204, Portland, Oregon, 1985. ACM Press, New York.

[BDK+91] G. Burosch, J. Demetrovics, G. O. H. Katona, D. J. Kleitman, and A. A. Sapozhenko. On the number of databases and closure operations. *TCS*, 78(2):377–381, 1991.

[Cam02] R. Camps. Transforming n-ary relationships to database schemas: An old and forgotten problem. Technical Report LSI-5-02R, Universitat Politècnica de Catalunya, 2002.

[Cod70] E. F. Codd. A relational model for large shared data banks. *CACM*, 13(6):197–204, 1970.

[DLM89] J. Demetrovics, L. O. Libkin, and I. B. Muchnik. Functional dependencies and the semilattice of closed classes. In *Proc. MFDBS'89, LNCS 364*, pages 136–147, 1989.

[HK01] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, Morgan Kaufmann Publishers, San Diego, USA, 2001.

[HN05] N. Habib and L. Nourine. The number of moore families on n=6. *Discrete Mathematics*, 294(3):291–296, 2005.

[Spy87] N. Spyratos. The partition model: A deductive data base model. *ACM Transactions on Database Systems*, 12(1):1–37, 1987.

[Tha87] B. Thalheim. Open problems in relational database theory. *Bull. EATCS*, 32:336 – 337, 1987.

[Tha00] B. Thalheim. *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin, 2000. See also http://www.informatik.tu-cottbus.de/∼thalheim/HERM.htm.

[Ull89] J. D. Ullman. *Principles of database and knowledge-base systems*. Computer Science Press, Rockville, MD, 1989.