

---

## Preface

Big data is a popular term used to describe the exponential growth, availability and use of information, both structured and unstructured. Much has been written on the big data trend and how it can serve as the basis for innovation, differentiation and growth.

According to International Data Corporation (IDC) (one of the premier global providers of market intelligence, advisory services, and events for the information technology, telecommunications and consumer technology markets), it is imperative that organizations and IT leaders focus on the ever-increasing volume, variety and velocity of information that forms big data. From Internet sources, available to all riders, here I briefly cite most of them:

- **Volume.** Many factors contribute to the increase in data volume—transaction-based data stored through the years, text data constantly streaming in from social media, increasing amounts of sensor data being collected, etc. In the past, excessive data volume created a storage issue. But with today's decreasing storage costs, other issues emerge, including how to determine relevance amidst the large volumes of data and how to create value from data that is relevant.
- **Variety.** Data today comes in all types of formats—from traditional databases to hierarchical data stores created by end users and OLAP systems, to text documents, email, meter-collected data, video, audio, stock ticker data and financial transactions.
- **Velocity.** According to Gartner, velocity means both how fast data is being produced and how fast the data must be processed to meet demand. Reacting quickly enough to deal with velocity is a challenge to most organizations.
- **Variability.** In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks. Daily, seasonal and event-triggered peak data loads can be challenging to manage—especially with social media involved.
- **Complexity.** When you deal with huge volumes of data, it comes from multiple sources. It is quite an undertaking to link, match, cleanse and transform data across systems. However, it is necessary to connect and correlate relationships, hierarchies and multiple data linkages or your data can quickly spiral out

of control. Data governance can help you determine how disparate data relates to common definitions and how to systematically integrate structured and unstructured data assets to produce high-quality information that is useful, appropriate and up-to-date.

Technologies today not only support the collection and storage of large amounts of data, they provide the ability to understand and take advantage of its full value, which helps organizations run more efficiently and profitably.

We can consider a Relational Database (RDB) as an unifying framework in which we can integrate all commercial databases and database structures or also unstructured data wrapped from different sources and used as relational tables. Thus, from the theoretical point of view, we can choose RDB as a general framework for data integration and resolve some of the issues above, namely volume, variety, variability and velocity, by using the existing Database Management System (DBMS) technologies.

Moreover, simpler forms of integration between different databases can be efficiently resolved by Data Federation technologies used for DBMS today.

More often, emergent problems related to the complexity (the necessity to connect and correlate relationships) in the systematic integration of data over hundreds and hundreds of databases need not only to consider more complex schema database mappings, but also an evolutionary graphical interface for a user in order to facilitate the management of such huge and complex systems.

Such results are possible only under a clear theoretical and *algebraic* framework (similar to the algebraic framework for RDB) which extends the standard RDB with more powerful features in order to manage the complex schema mappings (with, for example, merging and matching of databases, etc.). More work about Data Integration is given in pure logical framework (as in RDB where we use a subset of the First Order Logic (FOL)). However, unlike with the pure RDB logic, here we have to deal with a kind of Second Order Logic based on the tuple-generating dependencies (tgds). Consequently, we need to consider an ‘algebraization’ of this subclass of the Second Order Logic and to translate the declarative specifications of logic-based mapping between schemas into the algebraic graph-based framework (sketches) and, ultimately, to provide denotational and operational semantics of data integration inside a universal algebraic framework: the *category theory*.

The kind of algebraization used here is different from the Lindenbaum method (used, for example, to define Heyting algebras for the propositional intuitionistic logic (in Sect. 1.2), or used to obtain cylindric algebras for the FOL), in order to support the compositional properties of the inter-schema mapping.

In this framework, especially because of Big Data, we need to theoretically consider both the inductive and coinductive principles for databases and infinite databases as well. In this semantic framework of Big-Data integration, we have to investigate the properties of the basic **DB** category both with its topological properties.

Integration across heterogeneous data resources—some that might be considered “big data” and others not—presents formidable logistic as well as analytic challenges, but many researchers argue that such integrations are likely to represent the

most promising new frontiers in science [2, 5, 6, 10, 11]. This monograph is a synthesis of my personal research in this field that I developed from 2002 to 2013: this work presents a complete formal framework for these new frontiers in science.

Since the late 1960s, there has been considerable progress in understanding the algebraic semantics of logic and type theory, particularly because of the development of categorical analysis of most of the structures of interest to logicians. Although there have been other algebraic approaches to logic, none has been as far reaching in its aims and in its results as the categorical approach. From a fairly modest beginning, categorical logic has matured very nicely in the past four decades.

Categorical logic is a branch of category theory within mathematics, adjacent to mathematical logic but more notable for its connections to theoretical computer science [4]. In broad terms, categorical logic represents both syntax and semantics by a category, and an interpretation by a functor. The categorical framework provides a rich conceptual background for logical and type-theoretic constructions. The subject has been recognizable in these terms since around 1970.

This monograph presents a categorical logic (denotational semantics) for database schema mapping *based on views* in a very general framework for database-integration/exchange and peer-to-peer. The base database category **DB** (instead of traditional **Set** category), with objects instance-databases and with morphisms (mappings which are not simple functions) between them, is used at an *instance level* as a proper semantic domain for a database mappings based on a set of complex query computations.

The higher logical *schema level* of mappings between databases, usually written in some high expressive logical language (ex. [3, 7], GLAV (LAV and GAV), tuple generating dependency) can then be translated functorially into this base “computation” category.

Different from the Data-exchange settings, we are not interested in the ‘minimal’ instance  $B$  of the target schema  $\mathcal{B}$ , of a schema mapping  $\mathcal{M}_{AB} : \mathcal{A} \rightarrow \mathcal{B}$ . In our more general framework, we do not intend to determine ‘exactly’, ‘canonically’ or ‘completely’ the instance of the target schema  $\mathcal{B}$  (for a fixed instance  $A$  of the source schema  $\mathcal{A}$ ) just because our setting is more general and the target database is *only partially* determined by the source database  $\mathcal{A}$ . Another part of this target database can be, for example, determined by another database  $\mathcal{C}$  (that is not any of the ‘intermediate’ databases between  $\mathcal{A}$  and  $\mathcal{B}$ ), or by the software programs which update the information contained in this target database  $\mathcal{B}$ . In other words, the Data-exchange (and Data integration) settings are only special particular simpler cases of this *general framework* of database mappings where each database can be mapped from other sources, or maps its own information into other targets, and is to be locally updated as well.

The new approach based on the behavioral point of view for databases is assumed, and behavioral equivalences for databases and their mappings are established. The introduction of observations, which are computations without side-effects, defines the fundamental (from Universal algebra) monad endofunctor  $T$ , which is also the closure operator for objects and for morphisms such that the database lattice  $\langle Ob_{DB}, \preceq \rangle$  is an algebraic (complete and compact) lattice, where

$Ob_{DB}$  is a set of all objects (instance-database) of **DB** category and “ $\preceq$ ” is a pre-order relation between them. The join and meet operators of this database lattice are Merging and Matching database operators, respectively.

The resulting 2-category **DB** is symmetric (also a mapping is represented as an object (i.e., instance-database)) and hence the mappings between mappings are a 1-cell morphisms for all higher meta-theories. Moreover, each mapping is a homomorphism from a Kleisli monadic T-coalgebra into the cofree monadic T-coalgebra. The database category **DB** has nice properties: it is equal to its dual, complete and cocomplete, locally small and locally finitely presentable, and monoidal biclosed V-category enriched over itself. The monad derived from the endofunctor T is an enriched monad.

Generally, database mappings are not simply programs from values (i.e., relations) into computations (i.e., views) but an equivalence of computations because each mapping between any two databases  $A$  and  $B$  is symmetric and provides a duality property to the **DB** category. The denotational semantics of database mappings is given by morphisms of the Kleisli category **DB**<sub>T</sub> which may be “internalized” in **DB** category as “computations”. Special attention is devoted to a number of practical examples: query definition, query rewriting in the Database-integration environment, P2P mappings and their equivalent GAV translation.

The book is intended to be accessible to readers who have specialist knowledge of theoretical computer science or advanced mathematics (category theory), so it attempts to treat the important database mapping issues accurately and in depth. The book exposes the author’s original work on database mappings, its programming language (algebras) and denotational and operational semantics. The analysis method is constructed as a combination of technics from a kind of Second Order Logic, data modeling, (co)algebras and functorial categorial semantics.

Two primary audiences exist in the academic domain. First, the book can be used as a text for a graduate course in the Big Data Integration theory and methods within a database engineering methods curriculum, perhaps complementing another course on (co)algebras and category theory. This would be of interest to teachers of computer science, database programming languages and applications in category theory. Second, researches may be interested in methods of computer science used in databases and logics, and the original contributions: a category theory applied to the databases. The secondary audience I have in mind is the IT software engineers and, generally, people who work in the development of the database tools: the graph-based categorial formal framework for Big Data Integration is helpful in order to develop new graphic tools in Big Data Integration. In this book, a new approach to the database concepts developed from an observational equivalence based on views is presented. The main intuitive result of the obtained basic database category **DB**, more appropriate than the category **Set** used for categorial Lawvere’s theories, is to have the possibility of making synthetic representations of database mappings and queries over databases in a graphical form, such that all mapping (and query) arrows can be composed in order to obtain the complex database mapping diagrams. For example, for the P2P systems or the mappings between databases in complex data warehouses. Formally, it is possible to develop a graphic (sketch-based) tool for a

meta-mapping description of complex (and partial) mappings in various contexts with a formal mathematical background. A part of this book has been presented to several audiences at various conferences and seminars.

---

## Dependencies Between the Chapters

After the introduction, the book is divided into three parts. The first part is composed of Chaps. 2, 3 and 4, which is a nucleus of this theory with a number of practical examples. The second part, composed of Chaps. 5, 6 and 7, is dedicated to computational properties of the **DB** category, compared to the extensions of the Codd's SPRJU relational algebra  $\Sigma_R$  and Structured Query Language (SQL). It is demonstrated that the **DB** category, as a denotational semantics model for the schema mappings, is computationally equivalent to the  $\Sigma_{RE}$  relational algebra which is a complete extension of  $\Sigma_R$  with all update operations for the relations. Chapter 6 is then dedicated to define the abstract computational machine, the categorial RDB machine, able to support all **DB** computations by SQL embedding. The final sections are dedicated to categorial semantics for the database transactions in time-sharing DBMS. Based on the results in Chaps. 5 and 6, the final chapter of the second part, Chap. 7, then presents full operational semantics for database mappings (programs).

The third part, composed of Chaps. 8 and 9, is dedicated to more advanced theoretical issues about **DB** category: matching and merging operators (tensors) for databases, universal algebra considerations and algebraic lattice of the databases. It is demonstrated that the **DB** category is not a Cartesian Closed Category (CCC) and hence it is not an elementary topos. It is demonstrated that **DB** is monoidal biclosed, finitely complete and cocomplete, locally small and locally finitely presentable category with hom-objects ("exponentiations") and a subobject classifier.

Thus, **DB** is a weak monoidal topos and hence it does not correspond to propositional intuitionistic logic (as an elementary, or "standard" topos) but to one intermediate superintuitionistic logic with strictly more theorems than intuitionistic logic but less than the propositional logic. In fact, as in intuitionistic logic, it does not hold the excluded middle  $\phi \vee \neg\phi$ , rather the *weak* excluded middle  $\neg\phi \vee \neg\neg\phi$  is valid.

---

## Detailed Plan

1. Chapter 1 is a formal and short introduction to different topics and concepts: logics, (co)algebras, databases, schema mappings and category theory, in order to render this monograph more self-contained; this material will be widely used in the rest of this book. It is important also due to the fact that usually database experts do a lot with logics and relational algebras, but much less with programming languages (their denotational and operational semantics) and still much less with categorial semantics. For the experts in programming languages and category theory, having more information on FOL and its extensions used for the database theory will be useful.

2. In Chap. 2, the formal logical framework for the schema mappings is defined, based on the second-order tuple generating dependencies (SOTgds), with existentially quantified functional symbols. Each tgd is a material implication from the conjunctive formula (with relational symbols of a source schema, preceded with negation as well) into a particular relational symbol of the target schema. It provides a number of algorithms which transform these logical formulae into the algebraic structure based on the theory of R-operads. The schema database integrity constraints are transformed in a similar way so that both the schema mappings and schema integrity-constraints are formally represented by R-operads. Then the compositional properties are explored, in order to represent a database mapping system as a graph where the nodes are the database schemas and the arrows are the schema mappings or the integrity-constraints for schemas. This representation is used to define the database mapping sketches (small categories), based on the fact that each schema has an identity arrow (mapping) and that the mapping-arrows satisfy the associative law for the composition of them.

The algebraic theory of R-operads, presented in Sect. 2.4, represents these algebras in a non-standard way (with carrier set and the signature) because it is oriented to express the compositional properties, useful when formalizing the algebraic properties for a composition of database mappings and defining a categorical semantics for them. The standard algebraic characterization of R-operads, as a kind of relational algebras, will be presented in Chap. 4 in order to understand the relationship with the extensions of the Select–Project–Rename–Join–Union (SPRJU) Codd’s relational algebras. Each Tarski’s interpretation of logical formulae (SOTgds), used to specify the database mappings, results in the instance-database mappings composed of a set of particular functions between the source instance-database and the target instance-database. Thus, an interpretation of a database-mapping system may be formally represented as a functor from the sketch category (schema database graph) into a category where an object is an instance-database (i.e., a set of relational tables) and an arrow is a set of mapping functions. Section 2.5 is dedicated to the particular property for such a category, namely the duality property (based on category symmetry).

Thus, at the end of this chapter, we obtain a complete algebraization of the Second Order Logic based on SOTgds used for the logical (declarative) specification of schema mappings. The sketch category (a graph of schema mappings) represents the syntax of the database-programming language. A functor  $\alpha$ , derived from a specific Tarski’s interpretation of the logical schema database mapping system, represents the semantics of this programming language whose objects are instance-databases and a mapping is a set of functions between them. The formal *denotational semantics* of this programming language will be provided by a database category **DB** in Chap. 3, while the *operational semantics* of this programming language will be presented in Chap. 7.

3. Chapter 3 provides the basic results of this theory, including the definition of the **DB** category as a denotational semantics for the schema database mappings. The objects of this category are the instance-databases (composed of the relational tables and an empty relation  $\perp$ ) and every arrow is just a set of functions

(mapping-interpretations defined in Sect. 2.4.1 of Chap. 2) from the set of relations of the source object (a source database) into a particular relation of the target object (a target database). The power-view endofunctor  $T : \mathbf{DB} \rightarrow \mathbf{DB}$  is an extension of the power-view operation for a database to morphisms as well. For a given database instance  $A$ ,  $TA$  is the set of all views (which can be obtained by SPRJU statements) of this database  $A$ . The Data Federation and Data Separation operators for the databases and a partial ordering and the strong (behavioral) and weak equivalences for the databases are introduced.

4. In Chap. 4, the *categorical functorial semantics of database mappings* is defined also for the database integrity constraints. In Sect. 4.2, we present the applications of this theory to data integration/exchange systems with an example for query-rewriting in GAV data integration system with (foreign) key integrity constraints, based on a coalgebra semantics. In the final section, a fixpoint operator for an infinite canonical solution in data integration/exchange systems is defined.

With this chapter we conclude the first part of this book. It is intended for all readers because it contains the principal results of the data integration theory, with a minimal introduction of necessary concepts in schema mappings based on SOTgds, their algebraization resulting in the  $\mathbf{DB}$  category and the categorical semantics based on functors. A number of applications are given in order to obtain a clear view of these introduced concepts, especially for database experts who have not worked with categorical semantics.

5. In Chap. 5, we consider the extensions of Codd's SPRJU relational algebra  $\Sigma_R$  and their relationships with the internal algebra of the  $\mathbf{DB}$  category. Then we show that the computational power of the  $\mathbf{DB}$  category (used as denotational semantics for database-mapping programs) is equivalent to the  $\Sigma_{RE}$  relational algebra, which extends the  $\Sigma_R$  algebra with all update operations for relations, and which is implemented as SQL statements in the software programming. We introduce an "action" category  $\mathbf{RA}$  where each tree-term of the  $\Sigma_{RE}$  relational algebra is equivalently represented by a single path term (an arrow in this category) which, applied to its source object, returns its target object. The arrows of this action category will be represented as the Application Plans in the abstract categorical RDB machines, in Chap. 6, during the executions of the embedded SQL statements.
6. Chapter 6 is a continuation of Chap. 5 and is dedicated to computation systems and categorical RDB machines able to support all computations in the  $\mathbf{DB}$  category (by translations of the arrows of the action category  $\mathbf{RA}$ , represented by the Application Plans of the RDB machine, into the morphisms of the database category  $\mathbf{DB}$ ). The embedding of SQL into general purpose programs, synchronization process for execution of SQL statements as morphisms in the  $\mathbf{DB}$  category, and transaction recovery are presented in a unifying categorical framework. In particular, we consider the concurrent categorical RDB machines able to support the time-shared "parallel" execution of several user programs.
7. Chapter 7 provides a complete framework of the operational semantics for database-mapping programs, based on final coalgebraic semantics (dual of the initial algebraic semantics introduced in Chap. 5 for the syntax monads (programming languages), and completed in this chapter) of the database-mapping

programs. We introduce an observational comonad for the final coalgebra operational semantics and explain the duality for the database mapping programs: specification versus solution. The relationship between initial algebras (denotational semantics) and final coalgebras (operational semantics) and their semantic adequateness is then presented in the last Sect. 7.5.

- Thus, Chaps. 5, 6 and 7 present the second part of this book, dedicated to the syntax (specification) and semantics (solutions) of database-mapping programs.
8. The last part of this book begins with Chap. 8. In this chapter, we analyze advanced features of the **DB** category: matching and merging operators (tensors) for databases, present universal algebra considerations and algebraic lattice of the databases. It is demonstrated that the **DB** category is not a Cartesian Closed Category (CCC) and hence it is not an elementary topos, so that its computational capabilities are strictly inferior to those of typed  $\lambda$ -calculus (as more precisely demonstrated in Chap. 5). It is demonstrated that **DB** is a V-category enriched over itself. Finally, we present the inductive principle for objects and the coinductive principle for arrows in the **DB** category, and demonstrate that its “computation” Kleisly category is embedded into the **DB** category by a faithful forgetful functor.
  9. Chapter 9 considers the topological properties of the **DB** category: in the first group of sections, we show the Database metric space, its Subobject classifier, and demonstrate that **DB** is a *weak monoidal topos*. It is proven that **DB** is monoidal biclosed, finitely complete and cocomplete, locally small and locally finitely presentable category with hom-objects (“exponentiations”) and a subobject classifier. It is well known that the intuitionistic logic is a logic of an elementary (standard) topos. However, we obtain that **DB** is not an elementary but a weak monoidal topos. Consequently, in the second group of sections, we investigate which kind of logic corresponds to the **DB** weak monoidal topos. We obtain that in the specific case when the universe of database values is a finite set (thus, without Skolem constants which are introduced by existentially quantified functions in the SOTgds) this logic corresponds to the standard propositional logic. This is the case when the database-mapping system is completely specified by the FOL. However, in the case when we deal with incomplete information and hence we obtain the SOTgds with existentially quantified Skolem functions and our universe must include the infinite set of distinct Skolem constants (for recursive schema-mapping or schema integrity constraints), our logic is then an intermediate or superintuitionistic logic in which the weak excluded middle formula  $\neg\phi \vee \neg\neg\phi$  is valid. Thus, this weak monoidal topos of **DB** has more theorems than intuitionistic logic but less than the standard propositional logic.

---

## Acknowledgements

Although it contains mostly original material (some of it has not been published before), this book is largely based on the previous work of other people, especially in the framework of the First Order Logic, and could not have been written without that work. I would like to acknowledge the contribution of these authors with



a number of references to their important research papers. Also, many of the ideas contained are the result of personal interaction between the author and a number of his colleagues and friends. It would be impossible to acknowledge each of these contributions individually, but I would like to thank all the people who read all or parts of the manuscript and made useful comments and criticisms: I warmly thank Maurizio Lenzerini with whom I carried out most of my research work on data integration [1, 8, 9]. I warmly thank Giuseppe Longo who introduced me to category theory and Sergei Soloviev who supported me while writing my PhD thesis. I warmly thank Eugenio Moggi and Giuseppe Rosolini for their invitation to a seminar at DISI Computer Science, University of Genova, Italy, December 2003, and for a useful discussion that have offered me the opportunity to make some corrections and to improve an earlier version of this work. Also, I thank all the colleagues that I have been working with in several data integration projects, in particular Andrea Cali and Domenico Lembo. Thanks are also due to the various audiences who endured my seminars during the period when these ideas were being developed and who provided valuable feedback and occasionally asked hard questions.

---

## Notational Conventions

We use logical symbols both in our formal languages and in the metalanguage. The notation slightly differs: in classical propositional and FOL, we use  $\wedge_c, \vee_c, \Rightarrow_c, \neg_c$  (and  $\exists, \forall$  for the FOL quantifiers), while in intuitionistic,  $\wedge, \vee, \Rightarrow, \neg$  (in Heyting algebra, we use  $\multimap$  for the relative-pseudocomplement instead of  $\Rightarrow$ , but for different carrier sets of Heyting algebras we will use the special symbols as well). As the metasympol of conjunction, the symbol  $\&$  will be used.

In our terminology, we distinguish functions (graphs of functions) and maps. A (graph of) *function* from  $X$  to  $Y$  is a binary relation  $F \subseteq X \times Y$  (subset of the Cartesian product of the sets  $X$  and  $Y$ ) with domain  $A$  satisfying the functionality condition  $(x, y) \in F \& (x, z) \in F$  implies  $y = z$ , and the triple  $\langle f, X, Y \rangle$  is then called a *map* (or morphism in a category) from  $A$  to  $B$ , denoted by  $f : X \rightarrow Y$  as well. The composition of functions is denoted by  $g \cdot f$ , so that  $(g \cdot f)(x) = g(f(x))$ , while the composition of mappings (in category) by  $g \circ f$ .  $\mathcal{N}$  denotes the set of natural numbers.

We will use the symbol  $:=$  (or  $=_{\text{def}}$ , or simply  $=$ ) for definitions and, more often,  $\triangleq$  as well. For the equality we will use the standard symbol  $=$ , while for different equivalence relations we will employ the symbols  $\simeq, \approx, =$ , etc. In what follows, ‘iff’ means ‘if and only if’. Here is some other set-theoretic notation:

- $\mathcal{P}(X)$  denotes the power set of a set  $X$ , and  $X^n \triangleq \overbrace{X \times \cdots \times X}^n$  the  $n$ -ary Cartesian product, and  $Y^X$  denotes the set of all functions from  $X$  to  $Y$ ;
- We use  $\subseteq$  for inclusion,  $\subset$  for proper inclusion (we use  $\leq, \preceq, \sqsubseteq$  for partial orders), and  $X \subseteq_\omega Y$  denotes that  $X$  is a *finite* subset of an infinite set  $Y$ ;
- $R^{-1}$  is the converse of a binary relation  $R \subseteq X \times Y$  and  $\overline{R}$  is the complement of  $R$  (equal to  $(X \times Y) \setminus R$ , where  $\setminus$  is the set-difference operation);

- $id_X$  is the identity map on a set  $X$ .  $|X|$  denotes the cardinality of a set (or sequence)  $X$ ;
- For a set of elements  $x_1, \dots, x_n \in X$ , we denote by  $\mathbf{x}$  the sequence (or *tuple*)  $\langle x_1, \dots, x_n \rangle$ , and if  $n = 1$  simply by  $x_1$ , while for  $n = 0$  the empty tuple  $\langle \rangle$ . An  $n$ -ary relation  $R$ , with  $n = ar(R) \geq 1$ , is a set (also empty) of tuples  $\mathbf{x}_i$  with  $|\mathbf{x}_i| = n$ , with  $\langle \rangle \in R$  (the empty tuple is a tuple of every relation);
- By  $\pi_{\mathbf{K}}(R)$ , where  $\mathbf{K} = [i_1, \dots, i_n]$  is a sequence of indexes with  $n = |\mathbf{K}| \geq 1$ , we denote the projection of  $R$  with columns defined by ordering in  $\mathbf{K}$ . If  $|\mathbf{K}| = 1$ , we write simply  $\pi_i(R)$ ;
- Given two sequences  $\mathbf{x}$  and  $\mathbf{y}$ , we write  $\mathbf{x} \subseteq \mathbf{y}$  if every element in the list  $\mathbf{x}$  is an element in  $\mathbf{y}$  (not necessarily in the same position) as well, and by  $\mathbf{x}\&\mathbf{y}$  their concatenation;  $(\mathbf{x}, \mathbf{y})$  denotes a tuple  $\mathbf{x}\&\mathbf{y}$  composed of variables in  $\mathbf{x}$  and  $\mathbf{y}$ , while  $\langle \mathbf{x}, \mathbf{y} \rangle$  is a tuple of two tuples  $\mathbf{x}$  and  $\mathbf{y}$ .

A *relational symbol* (a predicate letter in FOL)  $r$  and its extension (relation table)  $R$  will be called often shortly as “relation” where it is clear from the context. If  $R$  is the extension of a relational symbol  $r$ , we write  $R = \|r\|$ .

---

## References

1. D. Beneventano, M. Lenzerini, F. Mandreoli, Z. Majkić, Techniques for query reformulation, query merging, and information reconciliation—part A. Semantic webs and agents in integrated economies, D3.2.A, IST-2001-34825 (2003)
2. M. Bohlouli, F. Schulz, L. Angelis, D. Pahor, I. Brandic, D. Atlan, R. Tate, Towards an integrated platform for Big Data analysis, in *Integration of Practice-Oriented Knowledge Technology: Trends and Prospectives* (Springer, Berlin, 2013), pp. 47–56
3. R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa, DATA exchange: semantics and query answering, in *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)* (2003), pp. 207–224
4. B. Jacobs, *Categorical Logic and Type Theory*. Studies in Logic and the Foundation of Mathematics, vol. 141 (Elsevier, Amsterdam, 1999)
5. M. Jones, M. Schildhauer, O. Reichman, S. Bowers, The new bioinformatics: integrating ecological data from the gene to the biosphere. *Annu. Rev. Ecol. Evol. Syst.* **37**(1), 519–544 (2006)
6. A. Labrinidis, H.V. Jagadish, Challenges and opportunities with Big Data. *Proc. VLDB* **5**(12), 2032–2033 (2012)
7. M. Lenzerini, Data integration: a theoretical perspective, in *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)* (2002), pp. 233–246
8. M. Lenzerini, Z. Majkić, First release of the system prototype for query management. Semantic webs and agents in integrated economies, D3.3, IST-2001-34825 (2003)
9. M. Lenzerini, Z. Majkić, General framework for query reformulation. Semantic webs and agents in integrated economies, D3.1, IST-2001-34825, February (2003)

10. T. Rabl, S.G. Villamor, M. Sadoghi, V.M. Mulero, H.A. Jacobsen, S.M. Mankovski, Solving Big Data challenges for enterprise application performance management. *Proc. VLDB* **5**(12), 1724–1735 (2012)
11. S. Shekhar, V. Gunturi, M. Evans, K. Yang, Spatial Big- Data challenges intersecting mobility and cloud computing, in *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access* (2012), pp. 1–6

Tallahassee, USA

Zoran Majkić

Big Data Integration Theory  
Theory and Methods of Database Mappings,  
Programming Languages, and Semantics

Majkić, Z.

2014, XX, 516 p. 170 illus., Hardcover

ISBN: 978-3-319-04155-1