Databases and Artificial Intelligence

Nicole Bidoit, Patrick Bosc, Laurence Cholvy, Olivier Pivert, Marie-Christine Rousset

Metadata of the chapter that will be visualized in SpringerLink

Book Title	A Guided Tour of Artificial Intelligence Research	
Series Title		
Chapter Title	Databases and Artificial Intelligence	
Copyright Year	2020	
Copyright HolderName	Springer Nature Switzerland AG	
Corresponding Author	Family Name	**Bidoit**
	Particle	
	Given Name	**Nicole**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	LRI, Université Paris Sud
	Address	Paris, France
	Email	Nicole.Bidoit@lri.fr
Author	Family Name	**Bosc**
	Particle	
	Given Name	**Patrick**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	IRISA
	Address	Rennes, France
	Email	Patrick.Bosc@irisa.fr
Author	Family Name	**Cholvy**
	Particle	
	Given Name	**Laurence**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	ONERA
	Address	Toulouse, France
	Email	Laurence.Cholvy@onera.fr
Author	Family Name	**Pivert**
	Particle	
	Given Name	**Olivier**
	Prefix	
	Suffix	

	Role	
	Division	
	Organization	IRISA
	Address	Rennes, France
	Email	Olivier.Pivert@irisa.fr
Author	Family Name	**Rousset**
	Particle	
	Given Name	**Marie-Christine**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	University Grenoble Alpes & Institut Universitaire de France, CNRS, Inria, Grenoble INP, LIG
	Address	38000, Grenoble, France
	Email	Marie-Christine.Rousset@imag.fr

Abstract	This chapter presents some noteworthy works which show the links between Databases and Artificial Intelligence. More precisely, after an introduction, Sect. 2 presents the seminal work on “logic and databases” which opened a wide research field at the intersection of databases and artificial intelligence. The main results concern the use of logic for database modeling. Then, in Sect. 3, we present different problems raised by integrity constraints and the way logic contributed to formalizing and solving them. In Sect. 4, we sum up some works related to queries with preferences. Section 5 finally focuses on the problematic of database integration.

Databases and Artificial Intelligence

Nicole Bidoit, Patrick Bosc, Laurence Cholvy, Olivier Pivert and Marie-Christine Rousset

Abstract This chapter presents some noteworthy works which show the links between Databases and Artificial Intelligence. More precisely, after an introduction, Sect. 2 presents the seminal work on "logic and databases" which opened a wide research field at the intersection of databases and artificial intelligence. The main results concern the use of logic for database modeling. Then, in Sect. 3, we present different problems raised by integrity constraints and the way logic contributed to formalizing and solving them. In Sect. 4, we sum up some works related to queries with preferences. Section 5 finally focuses on the problematic of database integration.

1 Introduction

Research in databases and artificial intelligence have been maintaining close relations for more than thirty years. "Logic and databases" was the first scientific field at the intersection of databases and artificial intelligence (Gallaire and Minker 1987; Gallaire et al. 1981; Reiter 1983; Gallaire et al. 1983, 1984). Its aim was to formalize

N. Bidoit (✉)
LRI, Université Paris Sud, Paris, France
e-mail: Nicole.Bidoit@lri.fr

P. Bosc · O. Pivert
IRISA, Rennes, France
e-mail: Patrick.Bosc@irisa.fr

O. Pivert
e-mail: Olivier.Pivert@irisa.fr

L. Cholvy
ONERA, Toulouse, France
e-mail: Laurence.Cholvy@onera.fr

M.-C. Rousset
University Grenoble Alpes & Institut Universitaire de France, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France
e-mail: Marie-Christine.Rousset@imag.fr

P. Marquis et al. (eds.), *A Guided Tour of Artificial Intelligence Research*,
https://doi.org/10.1007/978-3-030-06170-8_3

420055_1_En_3_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:25/4/2019 Pages: 26 Layout: T1-Standard

in logic some of the problems raised by databases. This approach has first met some difficulties in a community which did not clearly distinguish basic concepts used in databases from technological considerations. But its interest has gradually been truly appreciated. This research first focused on relational databases, then considered more complex information like incomplete information, deduction rules, dynamic integrity constraints, fuzzy information, legal information etc. This research also addressed new functionalities of databases like for instance, querying distributed databases, cooperative answers generation, preference-based queries answering or studying confidentiality of information.

Logic is one of the most useful formalisms in this area: first order logic, possibilistic logic (Dubois and Prade 2004), temporal logic, (de Amo and Bidoit 1993, 1995), epistemic logic (Reiter 1988; Demolombe and Jones 1996), deontic logic (Cuppens and Demolombe 1996; Carmo et al. 1997), situation calculus (Reiter 1993), description logic (Baader et al. 2003). But some other formalisms are also used, like for instance, fuzzy sets (Zadeh 1965) or CP-nets (Brafman and Domshlak 2004).

An exhaustive description of all the contributions at the intersection of databases and the artificial intelligence goes beyond the scope of this chapter. We will only address some of them. Section 2 sums up the seminal work of the “Logic and database” area which opened a wide research field at the intersection of databases and artificial intelligence. Section 3 deals with dynamic integrity constraints. Section 4 considers preference-based queries. Finally, Sect. 5 addresses the problem of database integration.

2 Modeling Relational Databases with Logic

2.1 Seminal Work

Reiter (1983) has been one of the first to promote the use of logic in the databases. His work aimed at using first order logic to model relational databases and describe their functionalities: complex information modeling, expressing queries and query evaluation, database updating... The use of logic has been motivated by the fact that this formal tool allows one to express sentences (formulas) and to reason based on these sentences. Reiter and his colleagues have shown that these two aspects exist in databases: one need to express information (data, constraints) and reason with them (queries must be answered, constraints must be checked...) Reiter has shown that modeling databases with logic can be done according to two different approaches: according to the model theory approach, a database instance is an interpretation of a particular first order language; according to the proof theory approach, a database instance is a set of first order formulas. In the following, we define a relational database with respect to the model theory approach.

Definition 1 A relational database is a triplet (L, I, IC) so that:

- L is a first order language corresponding to the database schema. It is defined as follows:
  - Any attribute value of the database is represented by a constant symbol of L. To simplify, the same symbol is used.
  - Any attribute domain T of the database is represented by an unary predicate symbol T, called type.
  - Any n-ary relation schema R of the database is modeled by a n-ary predicate symbol R.
  - The binary predicate for equality $=$ is introduced.
- $I = (D_I, i)$ is an interpretation of the language L corresponding to a state or an instance of the database. Its domain D_I and its interpretation function i are defined as follows:
  - D_I is isomorphic to the set of constant symbols of L. It is thus isomorphic to the set of attribute values of the database.
  - $i(=) = \{(a, a) : a \in D_I\}$. I.e., the predicate = is interpreted by the diagonal of D_I^2.
  - Any type T is interpreted by the subset of D_I which contains the constants associated with the values of the attribute domain T.
  - Any n-ary predicate R which represents a n-ary relation schema is interpreted by a set of elements of D_I^n corresponding to the tuples of the instance of the relation R in the database state.
- IC is a set of formulas of L called *integrity constraints*. They are defined by:
  - Any constraint on the states of the database (primary key, functional or inclusion dependency, ?) is represented by a formula in IC.
  - The formula $\forall x \quad T(x) \leftrightarrow (x = a_i^1) \vee ... \vee (x = a_i^n)$ belongs to IC, for any attribute domain $T = \{a^1...a^n\}$.
  - The formula $\forall x_1 ... \forall x_n \quad R(x_1, ..., x_n) \rightarrow T_1(x_1) \wedge ... \wedge T_n(x_n)$ belongs to IC for any n-ary relation schema R whose attribute domains are $T_1, ..., T_n$.

One will notice that, because of the simplification on the choice of the constants and their interpretation, the interpretation I is indeed, an Herbrand interpretation.

Definition 2 The database (R, I, IC) is *consistent* iff $\models_I IC$. I.e., the interpretation I satisfies IC or equivalently, I is a model of IC.

In these works, the only integrity constraints which can be modeled are those that can be expressed in first order logic. In Sect. 3, we will come back to the notion of integrity constraint. We will see that there are some other kinds of integrity constraints, called dynamic integrity constraints, whose expression needs the use of temporal logic.

As for database querying, logic has proved to be useful for query simplification, query equivalence etc. These results were provided for queries expressed in relational algebra which is one of the most popular language in databases. These results are based on the fact that any algebraic query can be reformulated as a first order formula as it is shown in the following:

Let DB be a relational database, Q be a query expressed in relational algebra and $answer(Q, DB)$ be the answer of Q when evaluated over DB. Let (R, I, IC) be the logical representation of DB. Then, there is a formula of L associated with Q, denoted $t(Q, x_1, ...x_n)$ and whose free variables are $x_1 ... x_n$, such that: answer $(Q, DB) = \{< d_1 ... d_n > \in D_I^n \ : \ \models_I Q(d_1 ... d_n)\}$.[1]

For instance, consider two binary relations $Employee(e : Person; d : Department)$ and $Phone(e : Person; n : num)$. The first one relates employees to the departments they belong to, and the second one associates employees to their telephone numbers. Consider the algebraic query Q: $\prod_n \ \sigma_{d=CS} \ (Employee(e, d) \bowtie Phone(e, n))$. It aims at retrieving the telephone numbers of the employees who belong to the computer-science department. Its translation in logic is: $t(Q, x) = \exists y(Employee(y, CS) \wedge Phone(y, x))$.

But, if any algebraic query can be reformulated as a logical formula, the reverse is not true. More precisely, it has been shown that some logical formulas do not correspond to any algebraic query. This is the case of the disjunction f $Employee(x, computer) \vee Employee(Sally, y)$ which aims to find the pairs of individuals (e, d) so that e is an employee of the computer science department and then d can be anything or conversely, d is the department $Sally$ belongs to and e can be anything. Expressing such a formula in relational algebra is impossible. Note that the "answer" $\{< e, d > \ : \ \models_I f\}$ may be an infinite set of pairs. Thus, the language of first order logic is, in some sense, more powerful than the relational algebra for expressing database queries. In the next section, we will see that it is even too powerful for expressing queries since it allows one to express queries which have no meaning in the context of information and databases modeling.

Let us come back to the consequences of the previous property. Since a relational database can be expressed in logic and any algebraic query can be expressed as a logical formula, some of the problems raised in the database context can be studied and solved in logic. For instance, showing that two algebraic queries Q and Q' are equivalent (i.e., they provide identical answers in any coherent database state) comes down to showing that $IC \models t(Q, x_1 ... x_n) \leftrightarrow t(Q', x_1 ... x_n)$ i.e., showing that $t(Q, x_1 ... x_n) \leftrightarrow t(Q', x_1 ... x_n)$ is a logical consequence of IC. In the same way, showing that the answer of an algebraic query Q is always empty comes down to showing that the set of formulas $IC \cup \ t(Q, x_1 ... x_n)$ is inconsistent. This has been used in the domain of *cooperative answering*.

[1] Remember that by convention, we take the same symbol to represent a constant and the individual which interprets it.

2.2 Domain-Independent Formulas

The previous section emphasized the fact that the language of first order logic can be used in the context of databases to model information, queries and integrity constraints. However, some logical formulas do not have a clear meaning and thus must be discarded. For instance, the formula $Employee(x, computer) \lor Employee(Sally, y)$ already discussed above, or the formula $\forall x \exists y Phone(x, y)$ are problematic, even if they are well-formed formulas. Indeed, the last formula means that the property of having a telephone number is universal and thus has no meaning since every individual satisfies it. In a database which manages employee identifiers, department identifiers, etc.... expressing such a formula as an integrity constraint is considered as a conceptual error. It would imply that any object, even a telephone number, has got a telephone number, which is a nonsense. Indeed, what is meant is "any employee has got a telephone number" which is written $\forall x \exists y (Employe(x) \rightarrow Phone(x, y))$. Now, the property of having a telephone number is restricted to employees.

Another example of a frequent error consists in modeling the query "who does not belong to the CS department ?" by the formula $\neg Department(x, CS)$. In a database which manages employee identifiers, department identifiers, etc.... the answer will necessarily contain all the telephone numbers, department identifiers etc. which obviously do not belong to the CS department. In fact, what is meant by this query is "who are the employees not belonging to the CS department ?" and must be modeled by $Employee(x) \land \neg Department(x, CS)$.

The only formulas modeling queries for database processing are the *domain-independent formulas* (Kuhns 1967). The formulas which have been pointed out above are not domain-independent. The valuation of domain-independent formulas remains the same when one changes the interpretation domain without modifying the interpretation of predicates. Domain-independent formulas are defined by:

Definition 3 *(Domain-independent formulas)* The formula $F(x_1, ..., x_n)$ is domain-independent iff for any pair of interpretations $I =< D_I, i >$ and $I^* =< D_I \cup \{*\}, i >$ where I^* differs from I by one domain element $*$, we have:

$$\{< d_1, ..., d_n >\in D_I^n :\models_I F(d_1, ..., d_n)\} = \{< d_1, ..., d_n >\in D_{I^*}^n :\models_{I^*} F(d_1, ..., d_n)\}.$$

Although domain-independent formulas characterize logic formulas meaningful as database queries, the class of domain-independent formulas turns out not to be decidable. Thus, there is no algorithm which proves that any formula, modeling an integrity constraint or a query, is domain-independent. Studies have been carried out in order to find decidable subsets of domain-independent formulas. Among them, one finds the class of evaluable formulas (Demolombe 1992), the class of range restricted formulas (Nicolas 1982) or the class of *Safe formulas* (Ullman 1980).

Let us mention here a different approach to solve the same issue and according to which formulas expressing semantic integrity constraints or queries are not restricted.

420055_1_En_3_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:25/4/2019 Pages: 26 Layout: T1-Standard

This approach rather modifies the semantic of the language so that the valuation domain is restricted to *active domains* i.e, the set of individuals which have an occurrence in the interpretation of one predicate or in the formula expressing the query or integrity constraint. For instance, consider two predicates R (binary), S (unary) and the interpretation $I =< D_I, i >$ shown below, supposing that D_I={a_1, a_2, ..., b_1, ...} is infinite:

R	
a_1	b_1
a_2	a_2

S
a_3
a_2

The active domain *adom(I)* of I is the finite set $\{a_1, a_2, a_3, b_1\}$. The first order formula $\neg S(x)$ is not a domain-independent formula as shown previously but the number of valuations $\nu(x) \in adom(I)$ such that $\models_\nu \neg S(x)$ is finite. It is $\{a_1, b_1\}$ which is the answer to the query $\neg S(x)$ over I according to the active domain semantics.

Among the strongest results in the theory of query languages, recalled in (Abiteboul et al. 1995), are those showing the equivalence between the four following languages:

- first order logic restricted to domain-independent formulas
- first order logic restricted to Range-restricted formulas
- first order logic whose semantic is restricted to active domain
- relational algebra.

These equivalences strengthen each solution provided to the initial problem and allows the use of any of them without loosing generality. For instance, using the "active domain" approach in database is quite common for simplicity reasons.

Finally, let us notice that even if these results are quite old, they remain of interest inn the context of information modeling and its validation. This issue arises in database and in artificial intelligence and can be captured by: how can we be sure that the formula intending to model a given piece of information, really represents it ? Identifying that the formula written to express some property is domain-dependent proves an conceptual error although, writing a domain-independent formula does not eliminate any modeling error.

3 Integrity Constraints

The relational model like most database models[2] is quite poor from a semantic point of view. It allows one to specify tables (relations) whose cells contain elementary values. The number of columns of the table and the values allowed in each column are part of the table specification. However, table description through the relational model, is unable to exclude specific value combination, neither does it enables the inverse that is to enforce conditioned value occurrence. In general, the relational

[2]The relational model has been chosen in the introduction but models such as non normalized, complex value data and semi-structured models are concerned as well.

model does not allow to capture complex properties nor general laws that data should verify in order to conform to the real world applications.

The relational model, like other data models, is enriched with mechanism allowing to complement the data structure specification of tables with properties related to the application domain. These properties which are metadata are called integrity constraints. Integrity constraints acquisition and management (maintenance) are fundamental in several respects: (1) as mentioned above, the key objective is to ensure data reliability that is their compliance with the application domain, (2) like typing in programming languages, integrity constraints have a powerful leverage effect for query and update optimization at the logical and physical level; constraints serve to model data and to efficiently manage data up to avoiding the evaluation of a query; for instance, based on the declared integrity constraints, one may statically identify that a query answer is empty.

Application evolution, from relational database to XML data systems, comes with the increased need to develop techniques ensuring data reliability and highly efficient management.

This section does not aim to address integrity constraint system features exhaustively (Abiteboul et al. 1995; Bidoit and Collet 2001), and even less to cover commercial systems. Our goal is to review some of the problems related to integrity constraints illustrating the link between database and artificial intelligence. The first part focuses on elementary notions and more specifically on first order logic formalization of integrity constraints. The second part is dedicated to dynamic integrity constraints and temporal logic.

3.1 Integrity Constraints and First Order Logic

We postpone for now the discussion on constraint types and focus on static integrity constraints. A static integrity constraint is a property, no matter how complex, which can be checked by a simple test on the database current state. For instance, the property stating that an employee is assigned to only one department, is a static constraint.

Classically, a constraint is specified by a closed first order formula. Why? Besides the relative simplicity that first order logic provides for expressing properties, most problems related to integrity constraints are directly translated in logical terms allowing one to reuse existing formal results and tools as well as to develop new ones. Here follows a broad overview of the most known and common problems (see (Abiteboul et al. 1995; Bidoit and Collet 2001) for an extensive presentation and bibliography).

Entailment. Integrity constraints are metadata. It is fundamental, for instance, in order to validate the database schema, to be able to answer the following question: given a set of integrity constraints $\mathscr{C}$, is there any other constraint which are enforced by $\mathscr{C}$? and what are these constraints? This decision problem is well-known as the entailment problem in first order logic. The entailment, denoted $\mathscr{C} \models c$, checks whether a formula c is true as soon as the set of formulas $\mathscr{C}$ satisfied. From a purely syntactic point of view, the problem comes to exhibit an inference system (axiomatization) used, when appropriate, to build a proof of c from the formulas in $\mathscr{C}$. Algorithmic and complexity issues of integrity constraint entailment have been investigated for specific classes of constraints called dependencies. The best known axiomatization is that of Armstrong for functional dependencies (Armstrong 1974). The frontier between logic and databases is drawn by the entailment complexity. Considering sub-classes of constraints such as acyclic, unary or tuple generating dependencies has been motivated by their good complexity properties as well as their relevance from the application point of view.

Coherence. Once constraints dedicated to a specific application domain have been specified, it is unavoidable to check consistency and to answer the following question: do data exist that satisfy these constraints? This problem is strongly related to satisfiability of a set of formulas which is known as undecidable. However satisfiability and consistency slightly differ: a set of formulas is satisfiable as soon as one model exists, even if this model is empty while a set of formulas is coherent if a non empty model exists for this set.

Semantic Optimization. Query optimization is a critical issue and traditionally its investigation combines two approaches. On the one hand, physical optimization makes use of the physical database schema (access paths like indexes) to generate efficient query execution code: integrity constraints like keys and foreign keys entail database index creation which foster query compilation. On the other hand, semantic query optimization takes place at an earlier stage by metadata based rewriting.[3] In extreme case, semantic optimization replaces query evaluation and produces the query answer avoiding data access. Example: the query extracting people having two partners while a constraint tells that every body has at most one partner.

Technics such as chase (Maier et al. 1979) for semantic optimization are among the most elegant ones. Formalizing both queries and constraints in first order logic allows one to use partial subsumption to "simplify" queries. Description logics have greatly contributed to semantic query optimization (Chakravarthy et al. 1990).

Description logics have extensively been used and contributed to semantic optimization (Hacid and Rigotti 1995; Bergamaschi et al. 1997; Calvanese et al. 1998; Beneventano et al. 2003) for their ability to provide a unique framework to express schemas, integrity constraints and queries.

Although it is impossible here to review all issues related to integrity constraints and leading to cross fertilization between artificial intelligence and databases, we ought to have a short discussion about integrity constraint maintenance methods.

Integrity constraint maintenance. Integrity constraints allow one to control the database evolution and thus checking database consistency arise essentially upon updates. But, when exactly? Choosing when constraint checking is activated leads to different classes of methods. The post update methods control and, if necessary,

[3]Functional dependencies help in a significant way the optimization of data sorting which arises when evaluating SQL group by, order by and distinct command (Simmen et al. 1996).

420055_1_En_3_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:25/4/2019 Pages: 26 Layout: T1-Standard

handle integrity violation through cancellation, repair or adaptation, after update execution: the efficiency of this optimistic and naive strategy relies on filtering the relevant constraints that are checked (relevant w.r.t. the updates) and also on developing incremental check. The pre-update methods are related to static analysis and takes on the challenge to predict, before executing the updates, the correctness of the result w.r.t. integrity constraints. These methods cannot be general. A dynamic variant of such strategy has been motivated by programming technics and introducing pre-condition enforcing valid update processing. Transaction schemas and active rules systems offer alternative solutions, often partial ones to integrity maintenance.

3.2 Dynamic Constraints: First Order and Temporal Logics

Whatever the type (static, dynamic, transaction), integrity constraints participate to database evolution control: changing data relies on these constraints in order to validate the changes and maintain data integrity/quality. To be checked, a transaction constraint needs to access both the database state before the update and that after. The constraint stating that salaries can only increase is an example of a transaction constraint. A dynamic constraint requires, in general, the whole state history of the database, that is the sequence of states from the creation of the database to the current state. The constraint stating that an employee cannot be reassigned to a department where she has been working in the past, is an example of a dynamic integrity constraint.

Dealing with dynamic constraints requires first to capture the notion of database history. We choose an abstract, simple model leaving aside a number of interesting problems such as concrete time measures, durations, calendar, problem induced by time granularity changes, multi-temporality (validity versus transaction), efficient storage of database history, etc. Dealing with abstract temporal or historical database is generally based on two equivalent simple temporal data representations.

On the one hand, the implicit approach considers a temporal database $\mathscr{I}$ over a schema (language) $\mathscr{R}$ as a sequence of static states $I_1, \ldots, I_n$ that is of interpretation of the language $\mathscr{R}$ as defined in 2. Each state I_{i+1} of the sequence has been obtained from an update over the previous state I_i. On the other hand, the explicit representation of a temporal database relies on data time stamping with time stamps being stored in the database as regular data. Time is assumed discrete and linear and the domain of the time stamp attribute is $\mathbb{N}$. Translating an implicit temporal database $\mathscr{I}$ into a time stamped instance uses an extension $\mathscr{R}^{est}$ of the schema $\mathscr{R}$ simply obtained by adding an attribute T to each relation schema R, leading to a schema R^{est}. Formally, the instance of R^{est}, denoted $I^{est}(R^{est})$, is given by $I^{est}(R^{est}) = \bigcup_{i=1}^{n}(I_i(R) \times \{i\})$.

In the implicit case, the query languages used to express dynamic or temporal integrity constraints are built from the linear temporal logic TL (Prior 1957; Emerson 1990; Chomicki and Toman 1998). Formulas of TL over a language $\mathscr{R}$ extend first order formulas with the following rules: if φ_1 and φ_2 are formulas then $\varphi_1 \, until \, \varphi_2$ et $\varphi_1 \, since \, \varphi_2$ are TL formulas.

420055_1_En_3_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:25/4/2019 Pages: 26 Layout: T1-Standard

A database history $\mathscr{I}$ satisfies a TL formula $\varphi(\mathbf{x})$ at time point $i \in [1, n]$, given a valuation ν of the free variables $\varphi(\mathbf{x})$, denoted $[\mathscr{I}, i, \nu] \models$, if the following holds:

- $[\mathscr{I}, i, \nu] \models \varphi_1(\mathbf{x_1})\ \mathit{until}\ \varphi_2(\mathbf{x_2})$ iff there exists $j > i$ such that $[\mathscr{I}, j, \nu] \models \varphi_2(\mathbf{x_2})$ and for each k such that $i < k < j$, $[\mathscr{I}, k, \nu] \models \varphi_1(\mathbf{x_1})$.
- $[\mathscr{I}, i, \nu] \models \varphi_1(\mathbf{x_1})\ \mathit{since}\ \varphi_2(\mathbf{x_2})$ iff there exists $j < i$ such that $[\mathscr{I}, j, \nu] \models \varphi_2(\mathbf{x_2})$ and for each k such that $i > k > j$, $[\mathscr{I}, k, \nu] \models \varphi_1(\mathbf{x_1})$.

Based on the temporal operators *until* and *since*, other operators may be derived such as *next*, *prev*, ...

In the explicit case, queries and constraints are expressed through first order logic, with the restrictions explained in Sect. 2, and by distinguishing two types of variables, data variables and temporal ones. The language obtained is thus a first order two-sorted logic, denoted TS–FO.

For instance, expressing that an employee cannot be reassigned in a department where she has been working in the past, is expressed by:

- using TL : $\forall e, d\ G(Employee(e, d) \rightarrow \neg(True\ Since\ Employee(e, d)))$ where G is the temporal modality "always".
- using TS–FO : $\forall t, \forall e, d\ (Employee(e, d, t) \rightarrow \neg(\exists t'\ (t' < t \wedge Employee(e, d, t))$ where t and t' are temporal variables whereas e and d are data variables.

The comparative study of the temporal query languages TL and TS–FO is probably one of the topics that led to rather unexpected results. The choice of explicit versus implicit representations of time has no impact at the level of data representation, however it has an impact on the language expressivity. As opposed to the results established by Gabbay (1980) and Kamp (1968) in the propositional case, comparing TL and TS–FO expressivity showed that:

1. the restriction of TL to the future *until*, *next* modalities is strictly less expressive than TL (Abiteboul et al. 1999);
2. TL is strictly less expressive than TS–FO (Abiteboul et al. 1999; Bidoit et al. 2004; Toman 2003).

This result has been proved using communication complexity on the one hand, and independently using Ehrenfeucht-Fraïssé games for the order invariant fragments of TL and TS–FO. For instance, the very simple property stating that there exists two distinct states for which employee assignments to departments are exactly the same, is invariant w.r.t. the time order; it is straightforward to express this property in TS–FO: $\exists t_1, t_2\ (\forall e, d\ (Employee(e, d) \leftrightarrow Employee(e, d)))$. However, this property cannot be expressed in TL.

These results have motivated a number of investigations aiming at extending TL to build an implicit temporal language as powerful as TS–FO : Wolper (1983) introduces an extension of TL based on regular expression; Toman (2003) proves that there is no temporal modality able to reach this goal; (Abiteboul et al. 1999; Herr 1997) propose temporal iterators and fixed-point operators (Vardi 1988; Bidoit and Amo 1999) studies adding the operator "now" and (Abiteboul et al. 1999; Bidoit and Objois 2009) provide a hierarchy of these languages w.r.t. to expressivity.

As for static constraints, we conclude this subsection by providing a few pointers to methods dedicated to dynamic constraint maintenance. Two kinds of methods have been investigated. The first ones are based on the hypothesis that the database history is fully stored and used for constraint checking leading to technics similar to those developed for static constraints. The second methods try to avoid the storage of the whole database evolution and instead enrich the current database state with data relevant to the constraint checking mechanism (Chomicki 1995; Chomicki and Toman 1995): each update entails auxiliary relation updates. The main issue here is to use as least auxiliary relations as possible. For a given set of constraints, the number of auxiliary relations is required to be fixed and their content should only depend on the database. The contribution of such methods resides in decreasing secondary memory consumption and also improving execution time. However these methods suffer from the fact that storage and time optimization are pre-determined by and for a given set of integrity constraints, excluding the ability afterwards to deal with (check and evaluate) other constraints or queries at all. Bidoit and Amo (1998) proposes to treat temporal constraint checking using refinement technics borrowed from program specification: given a set of temporal constraints viewed as an abstract specification, a set of parameterized transactions together with composition rules, viewed as a concrete specification, is generated. This method, which is not general, however allows one to deal with a large class of temporal constraints.

3.3 Concluding Remarks

To conclude, it is important to highlight that integrity constraint definition and maintenance is a research topic which is still active and will remain active for a long time because integrity constraints provide a way to fill the gap between semantically poor data models and real world applications, highly demanding w.r.t. to semantic issues. For instance, although not developed in this section, the semi-structured data model and the web data exchange model XML require the definition and verification of integrity constraints for improving the quality of data management, the accuracy of reasoning and for optimization purposes. Many research works (Davidson et al. 2007; Arenas 2009) have addressed these problems for the XML format: keys, reference and functional dependencies are classical constraints that are useful for XML applications; path constraints are "new" constraints linked to the XML data format (Buneman et al. 2001; Buneman et al. 2003; Fan and Siméon 2003) In this context too, logic and more precisely modal logics (Kripke 1963) have been investigated as they offer a unique and simple formalization of graph properties as well as powerful reasoning mechanisms for these structures: labelled graphs (or trees) are commonly used to represent XML data (Calvanese et al. 1999; Alechina et al. 2003; Demri 2003). Specifying schemas and constraints, more specifically reference constraints has been investigated in (Bidoit and Colazzo 2007; Bidoit and de Amo 1998).

4 Database Preferences Queries

4.1 Introduction

The last two decades have witnessed a growing interest in the expression of preferences in database queries. The motivations for extending database queries with preferences are manifold. First, it appeared desirable to provide users with more expressive query languages, capable of faithfully reflecting the user intentions. Secondly, introducing preferences into queries provides a basis for rank-ordering the answers, which is particularly helpful when the result of a query is large. Finally, when a classical query produces an empty result, a relaxed (thus less restrictive) version has more chance to be satisfied by some of the elements of the database.

The approaches that aim to integrate preferences inside database queries may be classified into two categories (Hadjali et al. 2011) according to whether they are of a quantitative or a qualitative nature (see chapter "Compact Representation of Preferences" of Volume 1). In the first family of approaches, preferences are expressed in a quantitative way by means of a monotonous *scoring function* (the global score is positively correlated to partial scores, and each of these is computed by a function of one or several attribute values). As the scoring function associates a numerical degree with each tuple, tuple t_1 is preferred to tuple t_2 if the score of t_1 is greater than the score of t_2. On the other hand, in qualitative approaches, preferences are defined by means of *binary preference relations*. These two families of approaches are presented hereafter through some of their most typical representatives.

4.2 Quantitative Approaches

4.2.1 Explicit Scores Attached to Entities

The approach proposed by Agrawal and Wimmers (2000) enables a user to express his/her preference for an entity, either by associating it with a score between 0 and 1, or by expressing a veto (using the symbol ♮) or an indifference statement (default case) related to this entity. An entity is represented by a tuple in which the value of a field either belongs to the domain of the corresponding attribute or is equal to * (symbol that stands for any domain value other than those specified in the query). In order to illustrate these notions, let us consider a relation *car* of schema (#*i*, *make*, *model*, *type*, *color*, *price*, ...) describing different vehicles. A user expressing the preferences {(⟨Renault, Clio, red⟩, 0.4), (⟨Renault, Clio, *⟩, ♮), (⟨Opel, Corsa, green⟩, ♮), (⟨Ford, Fiesta, white⟩, 0.8)} means that he/she has a strong preference for white Ford Fiestas, a much lower preference for red Renault Clios, and that he/she absolutely rejects green Opel Corsas as well as any Renault Clio that is not red. The approach also includes a generic operator that makes it possible to combine preferences from several users.

The approach proposed by Koutrika and Ioannidis (2004) follows the same general philosophy but extends (Agrawal and Wimmers 2000) by considering a more general format for user preference profiles. It also makes it possible to express negative preferences (“I do not like SUVs”) and preferences about the absence of values (“I prefer cars without ESP”).

4.2.2 Fuzzy-Set-Based Approach

As classical sets can be used for defining Boolean predicates, fuzzy sets (Zadeh 1965)—which aim to describe classes of objects whose boundaries are vague—can be associated with gradual predicates (see chapter “Representations of Uncertainty in Artificial Intelligence: Probability and Possibility” of Volume 1).

Generally speaking, atomic fuzzy predicates correspond to adjectives of the natural language such as *recent*, *big*, *fast*, etc. A fuzzy predicate P can be modeled by a function μ_P (usually of a triangular or trapezoidal shape) of one or several domains in the unit interval $[0,\ 1]$. The degree $\mu_P(x)$ represents the extent to which element x satisfies the gradual predicate P (or, equivalently, the extent to which x belongs to the fuzzy set whose membership function is μ_P). An atomic fuzzy predicate may also compare two attribute values by means of a gradual comparison operator such as “approximately equal” or “much greater than”.

It is possible to alter the semantics of a fuzzy predicate by means of a *modifier*, which is generally associated with an adverb of the natural language. For instance, the modified predicate *very expensive* is more restrictive than *expensive*, and *rather high* is less demanding than *high*. The semantics of the modified predicate $mod\ P$ (where mod is a fuzzy modifier) can be defined compositionally, and several approaches have been proposed to do so, among which $\mu_{mod\,P}(x) = \mu_P(x)^n$.

Atomic and modified predicates can take place in compound conditions which go far beyond those that can be expressed in a classical querying framework. Conjunction (resp. disjunction) is interpreted by means of a triangular norm (resp. conorm) $\top$ (resp. $\bot$), for instance the minimum or the product (resp. the maximum or the probabilistic sum). As for negation, it is modeled by: $\forall x,\ \mu_{\neg P}(x) = 1 - \mu_P(x)$.

Operators of weighted conjunction and disjunction can also be used to assign different weights to the predicates of a query.

The operations of relational algebra can be extended in a rather straightforward manner to fuzzy relations (i.e., to relations resulting from fuzzy queries, where tuples are assigned a membership degree) by considering fuzzy relations as fuzzy sets on the one hand, and by giving a gradual meaning to the operations whenever it appears appropriate. It is worth emphasizing that the fuzzy-set-based approach to preference queries provides a *compositional* framework, contrary to most of the other approaches (either quantitative or qualitative). The definitions of the extended relational operators can be found in Bosc et al. (1999). As an illustration, we give hereafter the definition of the fuzzy selection, where r denotes a (fuzzy or classical) relation and φ is a fuzzy predicate.

$$\mu_{\sigma_\varphi(r)}(x) = \top(\mu_r(x),\ \mu_\varphi(x))$$

where $\top$ denotes a triangular norm (for instance the minimum).

The language SQLf described in Bosc and Pivert (1995), Pivert and Bosc (2012) extends the SQL norm so as to authorize the expression of fuzzy queries.

The fuzzy-set-based approach has also been applied to the querying of multimedia databases in Fagin (1998).

4.2.3 Top-*k* Queries

In the top-*k* approach (Chaudhuri and Gravano 1999), the user specifies ideal values for certain attributes as well as the number k of answers (the best ones) that he/she wants to obtain. The distance between an attribute value and the ideal value is computed by means of a simple difference, after a normalization step which maps every domain to the unit interval $[0,\ 1]$. The global distance is computed by aggregating the elementary distances using a function which can be the minimum, the sum, or the Euclidean distance. The global score obtained by a tuple is the complement to 1 of its global distance to the ideal object specified in the query. The computation steps are as follows:

1. from the threshold k, the chosen aggregation function, and statistics about the content of the relation considered, a threshold α that will be applied to the global score is derived;
2. a Boolean query calculating the set of elements whose score is at least equal to α—or a superset of it—is built;
3. this query is evaluated and the global score attached to every answer is calculated;
4. if at least k tuples having a score at least equal to α have been obtained, the k best are returned to the user; otherwise, the procedure is executed again (starting from Step 2) using a lower value of α.

4.3 Qualitative Approaches

4.3.1 Pareto-Order-Based Approaches

In the last decade, many algorithms have been proposed for efficiently computing the non-dominated answers (in the sense of Pareto order) to a given preference query. Seen as points in a multidimensional space, these answers constitute a so-called *skyline*. A pioneering work in this domain is that by Börzsönyi et al. (2001). First let us recall the principle of Pareto-order-based preference queries.

Let $\{G_1, G_2, ..., G_n\}$ be a set of atomic partial preferences. We denote by $t \succ_{G_i} t'$ (resp. $t \succeq_{G_i} t'$) the statement "tuple t satisfies preference G_i better than (resp. at least as well as) tuple t'". In the sense of Pareto order, a tuple t dominates another tuple

t' if and only if $\forall i \in [1, n]$, $t \succeq_{G_i} t'$ and $\exists k \in [1, n]$, $t \succ_{G_k} t'$. In other words, t dominates t' if it is at least as good as t' w.r.t. every preference, and it is strictly better than t' w.r.t. at least one preference.

Clearly, the approach based on Pareto order does not require any commensurability assumption between the satisfaction levels associated with the different elementary preferences, contrary to the fuzzy-set-based approach for instance. As a consequence, some points of the skyline (i.e., some elements of the result) may perform very poorly w.r.t. some atomic conditions (whereas they can be excellent w.r.t. some others), and the skyline approach only provides a strict partial order whereas the fuzzy approach yields a complete preorder. Kießling (2002), Kießling and Köstler (2002) laid the foundations of a preference query model based on Pareto order for relational databases. A preference algebra including an operator called *winnow* has also been proposed by Chomicki (2003) so as to integrate formulas expressing user preferences inside a relational framework (and SQL). In a similar spirit, Torlone et Ciaccia (2002) have introduced an operator named *Best* that aims to return the non-dominated tuples of a relation.

In such an approach, when preferences concern multiple attributes, the risk of obtaining many incomparable tuples tends to get high. Several techniques have been proposed for defining an ordering between two tuples that are incomparable in the sense of Pareto order, by exploiting for instance: (i) the number of tuples that each of the considered ones dominate (notion of k-representativity introduced by Lin et al. (2007)), or (ii) an order between the attributes concerned by the preferences, see e.g. the notions of k-dominance defined by Chan et al. (2006a), and k-frequency proposed by the same authors (Chan et al. 2006b).

4.3.2 CP-nets

The use of the structure called CP-net (Conditional Preference Network) for modeling database preference queries has first been suggested by Brafman and Domshlak (2004)—but this preference approach was initially developed in Artificial Intelligence (Boutilier et al. 2004) (cf. chapter “Compact Representation of Preferences” of Volume 1). A CP-net is a graphical representation of statements expressing conditional preferences of type *ceteris paribus*. The underlying idea is that the preferences of the user generally express that, in a given context, a partially described state of affairs is strictly preferred to another partially described state of affairs, the two states being mutually exclusive, according to the *ceteris paribus* semantics, i.e., all other things being considered equal in the descriptions of the two states. Using a CP-net, a user can describe how his/her preferences on the values of a given variable depend on the values of other variables. For instance, a user may formulate the following statements:

s_1: I prefer SUVs to sedans;
s_2: as for SUVs, I prefer the make Ford to Chrysler;
s_3: as for sedans, I prefer the make Chrysler to Ford;
s_4: concerning Ford cars, I prefer the color black to white.

In the CP-net approach applied to database querying (Brafman and Domshlak 2004), a preference is represented by a binary relation over a relation schema (where the attributes are assumed to be binary). Let R be a relation schema; a preference query Q over R consists of a set $Q = \{s_1, ..., s_m\}$ of statements (usually between sub-tuples of R, according to the *ceteris paribus* semantics).

From Q, one may infer a set of preference relations $\{>_{CP}(1), \ldots, >_{CP}(m)\}$, from which one may derive a global preference relation $>_{CP}(Q)$ that defines a strict partial order on the tuples of R.

It is worth emphasizing that the *ceteris paribus* semantics is opposed to the so-called *totalitarian* semantics which is implicitly favored by the database community (including those who advocate an approach based on Pareto order). The totalitarian semantics means that when evaluating the preference clause of a query, one does not take into account the values of the attributes that do not appear in this clause. Obviously, with the *ceteris paribus* semantics, the number of incomparable tuples is in general much higher than with the totalitarian one.

4.3.3 Domain Linearization

The approach proposed in Georgiadis et al. (2008) considers preferences defined as preorders on relational attributes and their respective domains. Let us consider again a relation *car* of schema (#*i*, *make*, *model*, *type*, *color*, *price*, ...) describing vehicles. An example of preference query in the sense of (Georgiadis et al. 2008) is made of the following statements:

(1) I prefer Volkswagen to both Opel and Ford (P_1);
(2) I prefer the colors black and grey to white (P_2);
(3) I prefer the type sedan to coupe, and coupe to SUV (P_3);
(4) the make is as important as the type, whereas the combination make-type is more important than the color (P_4).

Such statements define binary preference relations: (1), (2) and (3) on attribute domains, (4) on the set of attributes. These relations are supposed to be reflexive and transitive, i.e., to be preorders. The authors propose a technique for linearizing the domains associated with these partial preorders (let us recall that a domain, in the sense of domain theory, is a partially ordered set). This way, one can build a sequence of blocks (i.e., an ordered partition) of the result of the query. In such a sequence, each block contains tuples that are incomparable in the sense of the user preferences. The first block contains the elements that are the most preferred, and in every other block, for every element, there exists an element that is more preferred in the preceding block.

The algorithms proposed in Georgiadis et al. (2008) compute the sequence of blocks that constitute the result of a preference query without building the order induced on the tuples themselves. The idea is to exploit the semantics of a preference expression for linearizing the Cartesian product of all the attribute values that appear in this expression. Concretely, one moves from a set of statements expressing partial preferences to a lattice of queries, then to a lattice of answers, and finally to a sequence of blocks that constitutes the result.

With respect to the approaches based on Pareto order, the originality of this technique lies in the use of partial (as opposed to strict) preorders for modeling independent positive preferences. This makes it possible to distinguish between the notion of "equally preferred tuples" on the one hand and "incomparable tuples" on the other hand.

4.3.4 Possibilistic-Logic-Based Approach

In Hadjali et al. (2011), present a preference query model based on possibilistic logic (Dubois and Prade 2004), (see chapter "Representations of Uncertainty in Artificial Intelligence: Probability and Possibility" of Volume 1), where the queries involve symbolic weights expressed on a linearly ordered scale.

For handling these weights, it is not necessary to give them a precise value, which leaves the user the freedom not to specify any default order on the priorities between the preferences (contrary to CP-nets where such an order is induced by the structure of the preference graph). However, the user may specify a partial order between the preferences.

In the case of binary preferences, the possibilistic encoding of the conditional preference "in context c, a is preferred to b" is a pair of possibilistic formulas: $\{(\neg c \vee a \vee b, 1), (\neg c \vee a,\ 1 - \alpha)\}$. Hence, if c is true, one must have a or b (which are the only possible choices), and in context c, it is somewhat imperative that a be true. This corresponds to a constraint of the form $N(\neg c \vee a) \geq 1 - \alpha$ where N measures the necessity of the event given as an argument; this expression is itself equivalent to $\Pi(\neg a | c) \leq \alpha$ where Π is the possibility measure dual to N.

This constraint expresses that the possibility *not to have a* is upper bounded by α, i.e., $\neg a$ is all the more impossible as α is small. To move from the scale of necessity degrees to a scale of satisfaction (or possibility) degrees, the authors use a scale reversal operator denoted by $1 - (.)$. The priority level $1 - (\alpha)$ associated with a preference is thus transformed into a satisfaction degree α when this preference is violated. Even if the values of the weights are unknown, a partial order between the different choices, founded on the operator *leximin* (Dubois et al. 1997), can be induced.

A parallel may be established between this approach and that based on fuzzy set theory where atomic conditions in a query may be assigned a weight reflecting their importance. These two approaches are in fact complementary and may be interfaced, which makes it possible to handle gradual (rather than binary) preferences on numerical attributes.

420055_1_En_3_Chapter ✓ TYPESET DISK LE ✓ CP Disp.:25/4/2019 Pages: 26 Layout: T1-Standard

4.4 Concluding Remarks

It is well known that scoring functions cannot model all preferences that are strict partial orders (Fishburn 1999), not even some that may appear in a natural way in database applications (Chomicki 2003). For instance, scoring functions cannot capture skyline queries (see Hadjali et al. 2011). However, the skyline approach, and more generally dominance-based approaches, have some notable drawbacks: they produce in general a large number of incomparable tuples, they suffer from dominance rigidity (there is no distinction between tuples that are dominated by far and those that are near to dominant tuples), and they focus on the “best” answers only whereas quantitative approaches yield a layered set of items. Let us also mention that qualitative approaches are rather limited when it comes to combining preferences while the fuzzy-set-based approach makes it possible to express a great variety of trade-offs between criteria due to the large range of connectives coming from fuzzy logic.

The aspects related to the implementation of these models, in particular query optimization, could not be dealt with here, due to space limitation, but they are of course crucial in a database context, where the volume of data to manage is in general very large. Some elements about this issue may be found e.g. in Pivert and Bosc (2012).

5 Database Integration

5.1 Motivations

The goal of data integration is to provide a uniform access to a set of autonomous and possibly heterogeneous data sources in a particular application domain. This is typically what we need when, for instance, querying the *deep web* that is composed of a plethora of databases accessible through Web forms. We would like to be able with a single query to find relevant data no matter which database provides it.

The goal of a mediator (Wiederhold 2002) on top of existing data sources is to give users the illusion that they interrogate a centralized and homogeneous database management system by providing a query interface based on a single global schema (also called mediated schema). In contrast to a standard database management system, a mediator does not contain any data, which remain stored in the different data sources according to a format and a schema specific to each data source, but contains abstract descriptions of those data in the form of views. The views describe the content of each data source in function of the mediated schema. Formally, a view is a query (i.e., a logical formula) defined over the relations of the mediated schema and identified by a name. For answering to user queries that are expressed using the relations of the mediated schema, the extensions of the relations in the queries are not available: only the extensions of views are known by the mediator. The problem

420055_1_En_3_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:25/4/2019 Pages: 26 Layout: T1-Standard

of answering queries asked to a mediator is thus formally equivalent to the problem of computing the answers from views extensions. This problem is harder than the problem of standard evaluation of a query for which we have the complete information on the extensions of the relations appearing in the query. The difficulty comes from the fact that the instances of the relations in the query must be inferred from the instances (or extensions) of the views and from the definitions of these views. Even in simple cases, one cannot infer all the instances of the query's relations, as it can be illustrated in the following example.

Example 1 Let us consider a mediated schema that contains a single binary relation $Reservation$ relying a person to the persons for whom s/he has made a reservation. Consider the query *Q(x,y) : Reservation(x, y)* asking all pairs of persons (x, y) such that the person x has made a reservation for the person y. Suppose that only three very specific databases are available for answering such a query :

- DB1, that can only provide persons that have made a reservation for themselves and for somebody else. The content of this database can be described by the view $V1$ defined by $V1(x) : Reservation(x, x) \wedge \exists y(y \neq x \wedge Reservation(x, y))$.
- DB2, that can only provide persons that have made reservations. The content of this database can be described by the view $V2$ defined by $V2(x) : \exists y Reservation(x, y)$.
- DB3, that can only provide persons for whom reservations have been made. The content of this database can be described by the view $V3$ defined by $V3(x) : \exists y Reservation(y, x)$.

Suppose that the extensions of these views are: $V1(a)$, $V2(a)$, $V2(b)$, $V3(c)$. They enable the entailment of the incomplete extension of the relation $Reservation$: $Reservation(a, a)$, $Reservation(a, ?)$, $Reservation(b, ?)$, $Reservation(?, c)$. The only precise answer that we can infer with certainty for the query Q is $< a, a >$. The other precise answers, such as $< a, c >$ for example, are possible but not certain.

5.2 Query Answering By Rewriting

The problem is to compute *all* the precise answers that are certain. An answer is precise if it is totally instantiated. An answer to a query is certain if it is part of the result of the evaluation of the query against all the extensions of the relations in the query that are compatible with the views extensions and definitions.

In the setting of mediator-based integration of distant data sources, the problem of query evaluation, that is already more complicated than the standard problem of query evaluation on top of a database as we have just explained it, is made even more complex by the fact that the data in the views extensions are not easily available. The cost of the transfer of these data into the mediator is prohibitive since they are distributed and stored in distant data sources. In addition, these data are very often evolving and volatile. This make impossible to base the computation of certain answers on reasoning on views extensions. The only resources available within the mediator are the views definitions. The computation of the answers can only be done by *rewriting* the query in terms of views. This consists in reformulating the input query into a union of queries built on the names of the views, called query rewritings in function of the views. Each of these rewritings, being a query using names of views only, can then be evaluated in a standard manner against the extensions of the views involved in the rewritings. More precisely, the rewritings represent the query plans enabling the extraction from the different data sources of the elements of answers that are relevant for computing the certain answers of the input query. Their concrete execution requires however software interfaces (called *wrappers*) between the mediator and the data sources.

Finding rewritings that are equivalent (modulo views definitions) to the input query is not always possible. In general, we merely compute (maximal) rewritings *subsumed* by the input query. A rewriting is subsumed by the input query if, by replacing in the body of the rewriting each view by its definition, we obtain a logical formula that logically implies the body of the input query. Because of this logical implication, a rewriting subsumed by the input query provides a query plan whose execution returns answers that are guaranteed to be relevant to the input query.

Given a query and a set of views, the problem of rewriting queries using views consist in determining if it is possible to compute the set of all rewritings that are maximally subsumed by the query.

Example 2 Consider a mediated schema allowing one to define queries on employees of a company using the following relations: *Employee(e:Person, d:Department)*, $Phone(e : Person, p : PhoneNumber)$, $Office(e : Person, b : RoomNumber)$. Let us suppose that the data is stored in two distinct databases DB1 and DB2 whose content is specified in function of the relations of the mediated schema using the following two views:

- $V1(e, b, d) : Office(e, b) \wedge Employee(e, d)$
- $V2(e, p) : Phone(e, p) \wedge Employee(e, \text{“}toy\text{”})$.

DB1 provides information on employees, their office number and their department. DB2 provides phone numbers of the employees of the toy department.

Let us consider the query: $Q(p, b) : Phone(\text{“}sally\text{”}, p) \wedge Office(\text{“}sally\text{”}, b)$ asking the phone and office numbers of Sally. The only rewriting that can be obtained for this query using the two views $V1$ and $V2$ is: $Q_v(p, b) : V2(\text{“}sally\text{”}, p) \wedge V1(\text{“}sally\text{”}, b, d)$.

It is worthwhile to notice that the execution of the query plan corresponding to this rewriting does not guarantee to return answers, for several reasons. First, if Sally is not a member of the toy department, the execution of the query plan will not bring any result. This is due to the incompleteness of the available data for the relations in the mediated schema, that is declared in the view definitions: the only way to obtain phone numbers is to use $V2$, but its definition specifies that $V2$ can only provide phone numbers for employees of the toy department. Another cause for incompleteness is related to the fact that, in absence of additional information, we do not know if the databases whose content is specified by views definitions are complete with respect to these definitions.

A view extension is complete if we can assume that it contains all the answers to the query defined by the view. For instance, stating the completeness of the $V2$ extension in the above example means that we have the guarantee that the database DB2 whose content is modeled by $V2$ definition contains effectively *all* the phone numbers of *all* the employees of the toy department. This completeness assumption is often too strong in the setting of information integration where it is reasonable to assume the soundness of views extensions but not their completeness. Stating that the $V2$ extension is sound (without being necessarily complete) means that DB2 contains phone numbers of employees of the toy department only, but not necessarily for all of them.

5.3 Decidability and Complexity

A lot of work (Beeri et al. 1997; Levy 2001; Abiteboul and Duschka 1998; Calvanese et al. 2000a, b; Goasdoué 2001) has been done on the decidability and the complexity of the problems of query rewriting using views and of answering queries using views, in function of the languages used for expressing respectively the queries, the views and the rewritings, and depending on the assumptions made on the views extensions. In particular, (Abiteboul and Duschka 1998; Calvanese et al. 2000a) shows the influence of the completeness assumption of the views extensions on the complexity of the problem of answering queries using views. It has been shown in Abiteboul and Duschka (1998) that under the soundness assumption on the views extensions, answering Datalog queries from extensions of views defined as conjunctive queries is polynomial (in data complexity), whereas this problem is co-NP-complete if the views extensions are assumed to be complete. If the views and the queries are expressed in Datalog, then in both cases (soundness and completeness of views extensions), the problem of answering queries using views is undecidable. These kinds of results have been extended in Calvanese et al. (2000a) to languages of queries and views belonging to the description logics family (Baader et al. 2003).

The problem of rewriting queries using views has been studied in (Beeri et al. 1997; Goasdoué 2001) when the languages for queries, views and rewritings belong to the CARIN (Levy and Rousset 1998) family that combines Datalog with description logics (see chapter "Reasoning with Ontologies" of Volume 1).

It has been shown in Calvanese et al. (2000b) that evaluating the rewriting of a query does not guarantee to find *all* the answers that can be obtained by evaluating the query on top of the views extensions, even if the rewriting is equivalent to the query modulo the views definitions. This shows an additional cause for the possible incompleteness of the answers, which is the limit of the expressive power of the language for specifying the rewritings. It is possible that a rewriting, defined in a language more expressive than the rewriting language imposed for modeling the allowed query plans, leads to more answers than any rewriting in the considered rewriting language.

Goasdoué (2001) provides a sufficient condition that guarantees to obtain by rewritings all the answers that it is possible to obtain by evaluating the query from views extensions. If the query has a finite number of maximal rewritings defined as conjunctive queries with inequalities, then the result of the evaluation of the query against the views extensions is exactly the union of the answers obtained by executing the query plans corresponding to the maximal rewritings. As a consequence of this condition, a mediator will be able to compute all the answers in time that is polynomial in the size of the data (even if it is exponential in the size of the queries and of the views definitions). This result has been applied to design and implement the PICSEL mediator (Goasdoué et al. 2000; Rousset et al. 2002) in collaboration with France Telecom R& D.

More recently, description logics have evolved towards the design of tractable fragments such as the DL-Lite family (Calvanese et al. 2007) with good computational properties for querying data through ontologies.

Ontologies are at the core of the Semantic Web (Berners-Lee et al. 2001). They provide a conceptual view of data and services available through the Web in order to facilitate their handling. Answering conjunctives queries over ontologies is central for implementing the Semantic Web. The DL-Lite family (Calvanese et al. 2007) has been specially designed to guarantee a polynomial data complexity for the problem of answering conjunctive queries over data constrained by lightweight ontologies. Reformulating the query in function of the constraints and axioms declared in the ontology is necessary for guaranteeing the completeness of the answers. The important point is that this reformulation step (just like rewriting the query using views) is a reasoning problem independent of the data.

A major result of (Calvanese et al. 2007) is that DL-Lite is one of the maximal subset of first-order logic for which the problem of answering queries on top of massive data in presence of logical constraints on the schema is *tractable*.

DL-Lite is a subset of the ontology web language OWL[4] recommended by the W3C and more precisely of the recent standard OWL2.[5] DL-Lite extends RDFS[6] with the possibility to declare disjoint classes and to express functionality constraints on relations. RDFS is the W3C standard to describe metadata on resources in Linked Data and the Semantic Web.

The results obtained for DL-Lite have been generalized to *decentralized* query rewriting using views in Abdallah et al. (2009). For scalability as well as for robustness and data privacy, it is indeed relevant to study a fully decentralized model of the Semantic Web seen as a huge peer-to-peer data and ontology management system.

[4]http://www.w3.org/2004/OWL/.

[5]http://www.w3.org/TR/owl2-overview/.

[6]http://www.w3.org/TR/rdf-schema/.

6 Conclusion

This chapter first presented the seminal work on "logic and databases" which opened a wide research field at the intersection of databases and artificial intelligence. Then it showed some links between the two areas by focusing on integrity constraints satisfaction, preference-based queries and database integration.

This chapter does not intend to present a complete overview of relations between databases and artificial intelligence. In particular, some recent extensions of databases require using artificial intelligence techniques. For instance, querying databases which stores uncertain data requires using techniques from uncertainty management (see chapters "Representations of Uncertainty in Artificial Intelligence: Probability and Possibility" and "Representations of Uncertainty in Artificial Intelligence: Beyond Probability and Possibility" of Volume 1); querying databases which stores inconsistent data requires using inconsistency-tolerant techniques (see chapter "Argumentation and Inconsistency-Tolerant Reasoning" of Volume 1) or information fusion techniques (see chapter "Belief Revision, Belief Merging and Information Fusion" of Volume 1).

References

Abdallah N, Goasdoué F, Rousset MC (2009) DL- LITE$_{\mathcal{R}}$ in the light of propositional logic for decentralized data management. In: International joint conference on artificial intelligence (IJCAI)

Abiteboul S, Duschka OM (1998) Complexity of answering queries using materialized views. In: ACM (ed) PODS '98. Proceedings of the seventeenth ACM SIG-SIGMOD-SIGART symposium on principles of database systems, ACM Press, New York, NY 10036, USA

Abiteboul S, Herr L, van den Bussche J (1999) Temporal connectives versus explicit timestamps to query temporal databases. J Comput Syst Sci 58(1):54–68

Abiteboul S, Hull R, Vianu V (1995) Foundations of databases. Addison-Wesley

Agrawal R, Wimmers E (2000) A framework for expressing and combining preferences. Proc SIGMOD 2000:297–306

Alechina N, Demri S, de Rijke M (2003) A modal perspective on path constraints. J Log Comput 13(6):939–956

Arenas M (2009) Xml integrity constraints. In: Encyclopedia of database systems. Springer, pp 3592–3597

Armstrong W (1974) Dependency structures of data base relationships. In: Proceedings of IFIP congress, North Holland, Amsterdam, pp 580–583

Baader F, Calvanese D, McGuinness D, Nardi D, Patel-Schneider PF (eds) (2003) The description logic handbook: theory, implementation, and applications. Cambridge University Press

Beeri C, Levy A, Rousset MC (1997) Rewriting queries using views in description logics, editor = ACM. In: PODS '97, Proceedings of the sixteenth ACM SIG-SIGMOD-SIGART symposium on principles of database systems, May 12–14, 1997. ACM Press, Tucson, Arizona, New York, NY 10036, USA

Beneventano D, Bergamaschi S, Sartori C (2003) Description logics for semantic query optimization in object-oriented database systems. ACM Trans Database Syst 28:1–50

Bergamaschi S, Sartori C, Beneventano D, Vincini M (1997) Odb-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In: AI*IA, pp 435–438

Berners-Lee T, Hendler J, OLassila (2001) The semantic web. Scientific American, p 279

Bidoit N, de Amo S (1998) A first step towards implementing dynamic algebraic dependences. Theor Comput Sci 190(2):115–149

Bidoit N, Amo SD (1998) A first step towards implementing dynamic algebraic dependences. TCS 190(2):115–149

Bidoit N, Colazzo D (2007) Testing xml constraint satisfiability. Electr Notes Theor Comput Sci 174(6):45–61

Bidoit N, Objois M (2009) Fixpoint and while temporal query languages. J Log Comput 19(2):369–404

Bidoit N, de Amo S, Segoufin L (2004) Order independent temporal properties. J Log Comput 14(2):277–298

Bidoit N, Amo SD (1999) Implicit temporal query languages: towards completeness. In: Proceedings of the 19th conference on foundations of software technology and theoretical computer science, pp 245–257

Bidoit N, Collet C (2001) Contraintes d'intégrité et règles actives. In: Bases de Données et Internet (Modèles, Langages, et systèmes). Hermès, pp 47–74

Bőrzsőnyi S, Kossmann D, Stocker K (2001) The skyline operator. In: Proceedings of the 17th IEEE international conference on data engineering, pp 421–430

Bosc P, Pivert O (1995) SQLf: a relational database language for fuzzy querying. IEEE Trans Fuzzy Syst 3(1):1–17

Bosc P, Buckles B, Petry F, Pivert O (1999) Fuzzy sets in approximate reasoning and information systems—the handbook of fuzzy sets series. Chap Fuzzy databases. Kluwer Academic Publishers, pp 403–468

Boutilier C, Brafman R, Domshlak C, Hoos H, Poole D (2004) CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. J Artif Intell Res (JAIR) 21:135–191

Brafman R, Domshlak C (2004) Database preference queries revisited TR2004-1934. Computing and information science, Tech Rep. Cornell University

Buneman P, Fan W, Weinstein S (2003) Interaction between path and type constraints. ACM Trans Comput Log 4(4):530–577

Buneman P, Davidson SB, Fan W, Hara CS, Tan WC (2001) Reasoning about keys for xml. In: DBPL, pp 133–148

Calvanese D, Giacomo GD, Lenzerini M (1999) Representing and reasoning on xml documents: a description logic approach. J Log Comput 9(3):295–318

Calvanese D, Giacomo GD, Lembo D, Lenzerini M, Rosati R (2007) Tractable reasoning and efficient query answering in description logics: the dl-lite family. J Autom Reason (JAR) 39(3):385–429

Calvanese D, De Giacomo G, Lenzerini M (2000a) Answering queries using views in description logics. In: Proceedings of AAAI 2000

Calvanese D, De Giacomo G, Lenzerini M, Vardi M (2000b) Answering regular path queries using views. In: Proceedings of ICDE 2000

Calvanese D, Lenzereni M, Nardi D (1998) Description logics for conceptual data modeling. In: Logics for databases and information systems. Kluwer

Carmo J, Demolombe R, Jones A (1997) Toward a uniform logical representation of different kinds of integrity constraints. In: Proceedings of ECSQARU-FAPR'97, LNAI 1244. Springer, pp 614–620

Chakravarthy U, Grant J, Minker J (1990) Logic-based approach to semantic query optimization. TODS 15(2):162–207

Chan C, Jagadish H, Tan K, Tung A, Zhang Z (2006) Finding k-dominant skylines in high dimensional space. Proc of SIGMOD 2006:503–514

Chan C, Jagadish H, Tan K, Tung A, Zhang Z (2006b) On high dimensional skylines. In: Proceedings of EDBT 2006, LNCS 3896, pp 478–495

Chaudhuri S, Gravano L (1999) Evaluating top-k selection queries. In: Proceedings of the 25th VLDB conference, pp 399–410

Chomicki J (1995) Efficient checking of temporal integrity constraints using bounded history encoding. ACM Trans Database Syst 20(2):149–186

Chomicki J (2003) Preference formulas in relational queries. ACM Trans Database Syst 28:1–40

Chomicki J, Toman D (1995) Implementing temporal integrity constraints using an active dbms. IEEE Trans Knowl Data Eng 7(4):566–582

Chomicki J, Toman D (1998) Temporal logic in information systems. In: Chap 3: Logic for databases and information systems, Kluwer Academic Publisher, pp 31–70

Cuppens F, Demolombe R (1996) A deontic logic for reasoning about confidentiality. In: Proceedings of 3rd international workshop on deontic logic in computer science (DEON'96)

Davidson SB, Fan W, Hara CS (2007) Propagating xml constraints to relations. J Comput Syst Sci 73(3):316–361

de Amo S, Bidoit N (1993) Contraintes dynamiques d'inclusion et schémas transactionnels. In: Neuvièmes Journées Bases de Données Avancées

de Amo S, Bidoit N (1995) A first step towards implementing dynamic algebraic dependencies. In: 893 L (ed) Proceedings of 5th ICDT

Demolombe R (1992) Syntactical characterization of a subset of domain independent formulas. J ACM 39

Demolombe R, Jones A (1996) Integrity constraints revisited. J Interes Group Pure Appl Log 4(3)

Demri S (2003) Modal logics for semistructured data. Invited talk at "Third workshop on methods for modalities (M4M-3)"

Dubois D, Prade H (2004) Possibilistic logic: a retrospective and prospective view. Fuzzy Sets Syst 144(1):3–23

Dubois D, Fargier H, Prade H (1997) Beyond min aggregation in multicriteria decision: (ordered) weighted min, discri-min, leximin. In: Yager R, Kacprzyk J (eds) The ordered weighted averaging operators—theory and applications. Kluwer Academic Publisher, pp 181–192

Emerson EA (1990) Temporal and modal logic. In: van Leeuwen J (ed) In: Handbook of theoretical computer science volume B: formal models and semantics. Elsevier

Fagin R (1998) Fuzzy queries in multimedia database systems. Proc of PODS 1998:1–10

Fan W, Siméon J (2003) Integrity constraints for xml. J Comput Syst Sci 66(1):254–291

Fishburn P (1999) Preference structures and their numerical representations. Theor Comput Sci 217(2):359–383

Gabbay DM, Pnueli A, Shelah S, Stavi J (1980) On the temporal basis of fairness. In: POPL, pp 163–173

Gallaire H, Minker J (1978) Logic and databases. Plenum

Gallaire H, Minker J, Nicolas J (1981) Advances in database theory, vol 1. Plenum

Gallaire H, Minker J, Nicolas J (1983) Advances in database theory, vol 21. Plenum

Gallaire H, Minker J, Nicolas JM (1984) Logic and databases: a deductive approach. ACM Surv 16(2)

Georgiadis P, Kapantaidakis I, Christophides V, Nguer E, Spyratos N (2008) Efficient rewriting algorithms for preference queries. Proc of ICDE 2008:1101–1110

Goasdoué F (2001) Réécriture de requêtes en termes de vues dans carin et intégration d'informations. PhD thesis, Université Paris Sud XI - Orsay

Goasdoué F, Lattes V, Rousset MC (2000) The use of carin language and algorithms for information integration: the picsel system. Int J Coop Inf Syst 9:383–401

Hacid MS, Rigotti C (1995) Combining resolution and classification for semantic query optimization. In: DOOD, pp 447–466

Hadjali A, Kaci S, Prade H (2011) Database preference queries—a possibilistic logic approach with symbolic priorities. Ann Math Artif Intell 63(3–4):357–383

Herr L (1997) Langages de requête pour les bases de données temporelles. PhD thesis, Université Paris-Sud 11

Kamp HW (1968) Tense logic and the theory of linear order. PhD thesis, University of California, Los Angeles

Kießling W (2002) Foundations of preferences in database systems. In: Proceedings of the 2002 VLDB conference, pp 311–322

Kießling W, Köstler G (2002) Preference SQL—design, implementation, experiences. In: Proceedings of the 2002 VLDB conference, pp 990–1001

Koutrika G, Ioannidis YE (2004) Personalization of queries based on user preferences. In: Bosi G, Brafman RI, Chomicki J, Kießling W (eds) In: Preferences, vol 04271 of Dagstuhl Seminar Proceedings. IBFI, Schloss Dagstuhl, Germany

Kripke S (1963) Semantical considerations on modal logic. Acta Philos Fenn 16:83–94

Kuhns J (1967) Answering questions by computers—a logical study. Rand Memo RM 5428 PR, Rand Corporation, Santa Monica, California

Levy A (2001) Answering queries using views: a survey. vLDB J

Levy A, Rousset MC (1998) Combining horn rules and description logics in carin. Artif Intell 101

Lin X, Yuan Y, Zhang Q, Zhang Y (2007) Selecting stars: the k most representative skyline operator. Proc of the ICDE 2007:86–95

Maier D, Mendelzon AO, Sagiv Y (1979) Testing implications of data dependencies. ACM Trans Database Syst 4(4):455–469

Nicolas JM (1982) Logic for improving integrity checking in relational databases. Acta Inform 18(3)

Pivert O, Bosc P (2012) Fuzzy preference queries to relational databases. Imperial College Press, London, UK

Prior A (1957) Time and modality. In: John Locke lectures for 1955–56. Oxford University Press

Reiter R (1983) Towards a logical reconstruction of relational database theory. In: On conceptual modelling: perspectives from artificial intelligence, databases and programming languages. Springer

Reiter R (1988) What should a database know. In: Proceedings of PODS

Reiter R (1993) Proving properties of states in the situation calculus. Artif Intell 64(2)

Rousset MC, Bidault A, Froidevaux C, Gagliardi H, Goasdoué F, Reynaud C, Safar B (2002) Construction de médiateurs pour intégrer des sources d'informations multiples et hétérogène: le projet picsel. Inf Interact Intell 2:9–58

Simmen D, Shekita E, Malkemus T (1996) Fundamental techniques for order optimization. In: Jagadish H, Mumick I (eds) Proceedings of the 1996 ACM SIGMOD international conference on management of data. Montreal, Quebec, Canada, June 4–6, 1996, ACM Press, pp 57–67

Toman D (2003) On incompleteness of multi-dimensional first-order temporal logics. In: Proceedings of the 10th international symposium on temporal representation and reasoning, pp 99–106

Torlone R, Ciaccia P (2002) Finding the best when it's a matter of preference. In: Proceedings of the 10th Italian national conference on advanced data base systems (SEBD 2002), pp 347–360

Ullman JD (1980) Principles of database systems. Computer Science Press

Vardi M (1988) A temporal fixpoint calculus. In: Proceedings of the 5th ACM symposium on principles of programming languages, pp 250–259

Wiederhold G (2002) Mediators in the architecture of future information systems. IEEE Comput

Wolper P (1983) Temporal logic can be more expressive. Inf Control 56(1–2):72–99

Zadeh L (1965) Fuzzy sets. Inf Control 8:338–353

MARKED PROOF

Please correct and return this set

Please use the proof correction marks shown below for all alterations and corrections. If you wish to return your proof by fax you should ensure that all amendments are written clearly in dark ink and are made well within the page margins.

Instruction to printer	*Textual mark*	*Marginal mark*
Leave unchanged	• • • under matter to remain	
Insert in text the matter indicated in the margin		New matter followed by or
Delete	through single character, rule or underline or through all characters to be deleted	or
Substitute character or substitute part of one or more word(s)	through letter or through characters	new character or new characters
Change to italics	under matter to be changed	
Change to capitals	under matter to be changed	
Change to small capitals	under matter to be changed	
Change to bold type	under matter to be changed	
Change to bold italic	under matter to be changed	
Change to lower case	Encircle matter to be changed	
Change italic to upright type	(As above)	
Change bold to non-bold type	(As above)	
Insert 'superior' character	through character or where required	or under character e.g. or
Insert 'inferior' character	(As above)	over character e.g.
Insert full stop	(As above)	
Insert comma	(As above)	
Insert single quotation marks	(As above)	or and/or or
Insert double quotation marks	(As above)	or and/or or
Insert hyphen	(As above)	
Start new paragraph		
No new paragraph		
Transpose		
Close up	linking characters	
Insert or substitute space between characters or words	through character or where required	
Reduce space between characters or words	between characters or words affected	