ISBN 978-83-60810-51-4

Adding rules into database systems

Zdenka Telnarová
University of Ostrava 30. dubna
22, 701 03 Ostrava, Czech
Republic
Email:zdenka.telnarova@osu.cz

Abstract—There are several types of rules playing an important role in database systems. As opposed to simple database systems where the only reasoning service is query answering, more advanced database systems offer number of advanced reasoning services such as deductive query answering based on deductive rules and active input processing based on active rules. In this article we discuss the concept of active and deductive rules and describe a simple model that uses both of them.

I. Theoretical background

As opposed on simple database systems where querying is based on First-Order-Logic (FOL) and query is represented by FOL formula, databases that offer deductive query answering are based on clause logic. Let us start with short reviewing of general knowledge about logic and logic programming and its use in rule based representation. Definitions are used from [1]

Definition 1. (Signature)
S is a signature if S is a four-tuple (P, F, r, C) where:
- *P is a set of predicate symbols (P_1, P_2, ...P_n),*
- *F is a set of function symbols (F_1, F_2, ...F_m),*
- *r is arity or degree of functions and relations. For each P_i respectively F_j, r(P_i) respectively r(F_j) is a non-zero natural number denoting the arity of P_i resp. F_j,*
- *C is a set of constant symbols.*

Definition 2. (Alphabet)
An alphabet Σ consists of the following symbols:
- *Signature S = (P, F, r, C).*
- *Collection of variables V.*
- *Operators: ¬ (negation), ∧ (conjunction), ∨ (disjunction), → (implication), ↔ (equivalence).*
- *Quantifiers: ∀ (forall),. ∃ (exists).*
- *Parentheses and punctuation symbols: (,) and ,.*

Definition 3. (Terms)
A term is defined inductively as follows:
- *Variable is term.*
- *Constant is term.*
- *If f is a function symbol (f ∈ F) with arity m and t_1, t_2, ...t_m are terms of Σ, then f(t_1, t_2, ..., t_m) is term of Σ.*

Definition 4. (Atom)
If p is predicate symbol with arity m and t_1, t_2, ...t_m are terms of Σ, then p(t_1, t_2, ..., t_m) is an atomic formula (or atom). An atomic formula is a formula and all occurrences of variables in an atomic formula is free.

Definition 5. (Formula)
A formula is defined as follows:
Atom is formula.
If H and G are formulas, then
- *¬ H is formula, the occurrence of variables in ¬ H is free(bound) if it is free (bound) in H,*
- *H ∧ G is formula, the occurrence of variables in H ∧ G is free(bound) if it is free (bound) in H or G,*
- *H ∨ G is formula, the occurrence of variables in H ∨ G is free(bound) if it is free (bound) in H or G,*
- *H→G is formula, the occurrence of variables in H→ G is free(bound) if it is free (bound) in H or G,*
- *H↔G is formula, the occurrence of variables in H↔ G is free(bound) if it is free (bound) in H or G.*

If H is formula and X is variable, then ∀XH and ∃XH are formulas. All occurrences of X are bound.

Definition 6. (Literal)
A literal L is an atom or negation of an atom.

Definition 7. (Clause)
A clause is a formula such as ∀X($L_1 \vee L_2 \vee ... \vee L_m$) where each L_i is a literal and X{X_1,X_2 ...X_n} are all the variables occurring in $L_1 \vee L_2 \vee ... \vee L_m$.

Definition 8. (Horn-clauses)
Horn-clauses have the form:

. $\forall X_1, X_2, ...X_m.L1 \wedge L_2 \wedge .... \wedge L_n \rightarrow L$
where L, L_1, L_2, ...L_n are literals and X_1, X_2, ...X_m are all variables having free occurrences in L, L_1, L_2, ...L_n.

Often Horn-clauses are written as follows:

$L \leftarrow L_1 \wedge L_2 \wedge .... \wedge L_n$

II. Deductive Rules

Conventional database systems do not work with deductive rules and its partial function is exercised by queries. In advanced database systems the rules are concepts used in order to obtain information from the database in more effective way using declaration languages. In addition to it, the rules are used as means for maintaining database consistency or mapping abstract data types (ADT) into relational tables. From a formal standpoint, the rules are declarative expressions. By their evaluation (or an appropriate interpretation) it is possible to obtain additional information from the database.

Deductive rules can be used to:

• Express subsumption relationships between concepts, which is used in object-relational databases, where deductive rules describe mapping ADT into relationship between tables represented by corresponding sets of atomic sentences.

• Define intensional predicates, which are used for derivation of intensional concepts.

• Express default rules and other types of heuristic knowledge.

• Represent causal relationalship between causes and effects.

Deductive rule is based on Horn-clauses and the expression is in the form:

Conclusion ← Premise,

where conclusion is an atomic formula (head of the rule) and premise is a formula (body of the rule). Theory of deductive rules is based on deductive axioms; deductive axiom is a rule by which we are able to deduce additional facts from the given facts.

In terms of analysis, design and implementation of information system we can use deductive rules for derivation of classes/relations, derivation of attributes, views, for design of query, for expressing integrity constraint.

III. Deductive rules for data derivation

As it has already been mentioned, deductive rules can serve as a powerful tool for data derivation on the basis of extensional data stored in the database and for defining of requirement for database consistency. Creator of database system should weigh which classes/relations and attributes must be the part of extensional database, and which can be derived.

All the examples in this paper are written in Chimera language [2]. Chimera is a novel database model and language which has been designed as a joint conceptual interface of the IDEA project, a major European cooperation initiative aiming at the integration of object-oriented, active and deductive database technology. The most remarkable characteristic of Chimera is the fact that fully developed rule languages for both active and deductive rules have been integrated in an object-oriented context.

I. Rules for derivation of attributes

Concrete examples will be focused on students environment. Let us have, for example, a class Student:

```
Individual Student in Class with
attribute
  id_num : String,
  name : String,
  year : Integer,
  st_results : Results
End
```

where Results are also a class:

```
Individual Results in Class with
attribute
  semester : Integer,
  course: String,
  branch: String,
  credits_requested: Integer,
  credits_gained: Integer
End
```

On the basis of in this way defined extensional part of the database can be derived an attribute credits_in_semester, which is the attribute of class Student and can be expressed on the basis of knowledge of attributes semester and credits_gained, which are attributes of class Results; linking of these two classes is ensured by the attribute st_results:

```
Individual Student with
attribute
credits_in_semester : Integer derived
End
```

For calculation of the number of credits for particular semesters we will use function Number_of_credits, whose input parameter will be semester and output parameter will be the number of gained credits:

```
Define external formula
Number_of_credits (in semester:
integer, out I: integer)
End
```

This formula will be used for defining of deductive rule to derive the attribute credits_in_semester:

```
Define implementation for Student with
attribute
Self.credits_in_semester =  I←
integer(I), Number_of_credit
(Self.st_results.semester, I)
End
```

2. *Rules for deduction of recursive attributes*

For derivation of the attribute `prerequisites` a recursive deductive rule for the transitive closing of relation will be applied (the attribute `prerequisites` is a set of all courses that have to have been completed by a student so that they could enter their names for a given course):

```
Define implementation for Course
attribute
X in Self.prerequisites ← Course(X), X
in Self.requisite;
X in Self.prerequisites← Course(X), X
in Self.requisite.prerequisites
End
```

for a class `Course`:

```
Individual Course in Class with
attribute
  course_no: Integer;
  c_name: String;
  g_member: Group;
  requisite : Course
End
```

3. *Rules for derivation* **of classes**

On the basis of concrete values of the given attributes it is possible to define classes. Definitions of such classes are part of intensional database. Example can be a class `Inf_branch_student` that represents all students for which the value of attribute branch = Informatics.

```
Define implementation for class Student
population
Inf_branch_student(X) ←  Student(X),
X.branch = ,Informatics´
End
```

4. *Rules for inversion relation*

Let us have such classes `Course` and `Group` so that there is an inversion relation between them. Attribute `g_member` of the class `Group` and attribute `g_member` of the class `Course` are then inversion attributes. From the view of database integrity maintenance it is more convenient to define one of these attributes as derivated by applying a deductive rule:

```
Define implementation for Course
attribute
X in Self.g_member  ←  Group(X),
X.g_member= Self
End
```

Class `Group` will be defined:

```
Individual Group in Class with
attribute
  group_no: Integer;
  students: Student;
  g_member: Course
End
```

IV. Rules for Defining of Integrity Constraints

Deductive rules can serve for defining of integrity constraints both in a fix format and in so called generic format. The fix format usually concerns constraints of the type not null, primary key, unique attribute, inverse attribute and referential integrity. These integrity constraints are furthermore supported by conventional database systems. There are also integrity constraints in generic format that depend on a specified application domain. These constraints can be supported in their full generality only by new generations of database systems.

An integrity constraint is a first-order logic (FOL) formula that has to be a theorem of each database during which the constraint is considered valid. Deductive rules are formally a subset of the integrity constraints. A database satisfied the constraints when the rule is „false“ and does not generate any binding for the rule´s head. The database is not consistent when the rule is „true“.

1. *Targeted constraints*

Targeted constraint is unary predicate on relation/class, which is the name of the head of constraint (name is represented by unary predicate symbol). Argument of the formula is one and it is variable (for example: Self in Chimera or Oracle, arbitrary in Telos). Instances computing from the constraint are the violate constraint instances and they represent the values of the variable:

Constraint_name ← formula,
where formula consists of one relation/class.

2. *Untargeted constraints*

The main idea is the same. When untargeted constraints generate some bindings, the integrity of database is violated.

Untargeted constraints refer to several attributes from different relations/classes:

Constraint_name (argument1, argument2,...) ← formula,
where formula consists of several relations/classes.

3. *Dynamic constraints*

Dynamic constraints are important for monitoring state changes (past database state / current database state). Implementation of dynamic constraints have to manage denotation the past state of the target instances/objects.

4. *Referential integrity constraint*

Referential integrity is established between two relations/classes and it means that referencing instance/object does not exist without being linked to a referenced instance/object. Head of the rule is the name of the constraint and is represented by unary predicate. Instances/objects violated referential integrity are binding with the head.

Examples of integrity constraints:

- The study year of the student is an integral number between l - 5:

```
Add constraint error_year
for Student
as
error_year(Self) ←   Self.year < 1;
error_year(Self)  ←  Self.year > 5
End
```

- requested_credits >= gained_credits

```
Add  constraint error_credits
for Results
as
error_credits(Self)←Self.credits_gained
>= Self.credits_requested
End
```

V. Active rules

Many of the activities that are in conventional database system coded in each or every application program to support data management policies can be in advanced database systems "abstracted" and assigned to active rules. The trend in application design, which leads to modeling application semantics by means of deductive and active rules, contributes to achieving knowledge independence. The advantage of this approach lies in effective modification of the management policy by modifying rules instead of application programs.

Active rules can be used to:

- Integrity constraint maintenance.
- Data derivation and in particular data replication.
- Version maintenance, security administration, event tracking.
- Implement alerts, when the action consists of giving a message without changing the database content.
- Implement business rules, performing a part of the business computation that is normally part of application program.

In this article we mention only integrity constraint and business rules. We also add some examples.

1. Managing integrity constraint by means of active rules

At the simplest level constraint can be declaratively expressed by deductive rules, for example value of attribute `semester` of the class `Result` is positive number.

```
Add  constraint error_semester
for Result
as
error_semester(Self) ←   Self.semester
< = 0
End
```

This form of integrity constraint maintenance requires a sophistical integrity maintenance which has for example deductive databases.

In other level we can express constraint as trigger activated by specified event. If a violation is detected the trigger issues rollback command. These triggers are called abort rules. This approach has disadvantage in causing frequent rollbacks of the transactions.

Transcription of the declarative deductive rule mentioned above to the trigger (active rule) follows.

```
Define trigger error_semester
for Results
events create, modify (semester)
condition Self.semester < = 0
action rollback
End
```

To avoid `aborting transactions when constraint violation is occurred we can in action part use data manipulation command (for example to assign value 100 to the attribute semester)`.

```
Define trigger error_semester
for Results
events create, modify (semester)
condition Self.semester < = 0
action modify (semester, 100)
End
```

Rules for referential integrity

Active rules for referential integrity can be composed as follows:

Event is: delete from the referenced class.

Condition determine: whether an instance of the referencing class has become an orphan.

Action can be: either delete (cascade policy) or rollback the transaction (restrict policy).

We reuse the classes `Student` (referenced class) `and Results` (referencing class)

```
Individual Student in Class with
attribute
  id_num : String,
  name : String,
  year : Integer,
  st_results : Results
End

Individual Results in Class with
attribute
  semester : Integer,
  course: String,
  branch: String,
  credits_requested: Integer,
  credits_gained: Integer
End
```

A trigger implementing restricts policy is:

```
Define trigger delete_Student_restrict
for Results
event delete
condition occured(delete, Self),
Student(S), Self = old(S.st_results)
action rollback
End
```

A trigger implementing cascade policy is:

```
Define trigger delete_Student_cascade
for Results
event delete
condition occured(delete, Self),
Student(S), Self = old(S.st_results)
action delete(Student, S)
End
```

2. *Business rules*

Business rules respond to application needs. They model the reaction to events which occur in the real world. Business rules are primarily an outcome of the understanding of the business process. In particular, business rules model part of business process which is common to all applications and can be abstracted for all in the form of active rules.

Creating business rules by active rules has overall design strategy [2]:

Identify application tasks for active rules. Associate each task with the condition under which the task should be executed. Give a simple description of the task in the form “if condition, then action”.

Detect for each task the events that cause the task to be executed.

Generate active rules responding to the events associated with the task.

VI. Conclusion

Active and deductive rules in database systems score a fast progress at present. It is mainly because of their ability to model reality more precisely, to process complex data more efficiently and to ensure better the independence of data both on users´ applications and on their implementation. In connection with deductive rules in database systems we talk about a so called knowledge independence that can be considered as another step to flexibility of databases and their applications. Therefore, the field of analysis and design of information systems with a database by employing active and deductive rules approaches is certainly a topical theme.

References

[1] Chomicki, J., Saake, G.: Logics for databses and information systems. Kluwer, 1998

[2] Ceri, S., Fraternali, P.: Database Applications with objects and rules. Addison-Wesley, 1997 1. S. Abiteboul, R. Hull, V. Vianu: Foundations of Databases. Addison-Wesley Pubs. Co., 1995

[3] Ullman, D. J.: Principles of Database and Knowledge-Base Systems. Volume I, II. Computer Sc. Press, 1988

[4] Jarke, M., Staudt, M.: An Application Perspective to Deductive Object Bases. Proc. ACM-SIGMOND Workshop on Combining Declarative and Object-Oriented Databases, Washington D.C., May 1993

[5] Wagner, G.: Foundations of Knowledge Systems with Applications to Databases and Agents. Kluwer Academic Publishers, 1998

[6] Zaniolo, C., Ceri, S., Faloutsos, Ch., Snodgrass, R.T, Subrahmanian, V.S., Zicari, R.: Advanced Database Systems. Morgan Kaufmann Publishers, Inc., 1997

[7] Telnarova, Z.: Design Principles and Implementation of Deductive Rules. ISBN 80-7042-795-7, ICTE 2000, Roznov, pp 122-125

[8] "From Data to Classification Rules and Actions", Z. Ras, A. Dardzinska, International Journal of Intelligent Systems, Wiley, Vol. 26, Issue 6, 2011, 572-590
[http://www.cs.uncc.edu/~ras/Ras-Aga-IJIS.pdf]

[9] "Mining actionable patterns by role models", K. Wang, Y. Jiang, A. Tuzhilin, Proceedings of the 22nd International Conference on Data Engineering, 2006, 16-26
[http://www.cs.sfu.ca/~wangk/pub/icde06.pdf]