Expert Systems With Applications 97 (2018) 106–116

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Relational calculi in a deductive system☆

Fernando Sáenz-Pérez

Faculty of Computer Science Complutense University of Madrid Madrid 28040, Spain

ARTICLE INFO

Article history:
Received 21 December 2016
Revised 19 October 2017
Accepted 4 December 2017
Available online 7 December 2017

Keywords:
Tuple relational calculus
Domain relational calculus
Relational algebra
SQL
Datalog
Database systems

ABSTRACT

This work describes the addition of relational calculus languages in the deductive database system DES. Based on first-order logic, such languages admit a clean logical reading of queries, providing truly declarativeness, in contrast to other languages based on logic such as Prolog (a classical language used to build expert systems). Interesting properties as termination (for finite relations) and recursion are ensured because the DES deductive engine is used for solving relational calculus queries. Recursion in particular opens a brand new ream of applications (social networks, data warehouses, ...) for relational calculus languages which were unmanageable up to now. Since the DES system was targeted at teaching, we have also make a special emphasis on providing a practical system for students by providing appropriate syntax error feedback in a system supporting different languages (including relational algebra, SQL and Datalog).

1. Introduction

An expert system can be built with an inference engine which processes rules to deduce new facts in addition to the already stored ones in a knowledge base. Rules can be as simple as if-then rules or more elaborated forms based on First-Order Logic (FOL). Systems implementing Prolog (a traditional language for building expert systems (Merritt, 2001)) are examples of such FOL inference engines with support for a dynamic knowledge base. Despite the acknowledged declarativeness of such systems, they however present some drawbacks to build high-abstraction-level expert systems, including the following: First, the declarative level of rules is lowered because the order of both rules and conjunctions in a rule affects semantics. Second, non-logical construct as the *cut* break the logical reading of rules, making the development of logical systems tricking and prone to errors. Third, they are geared towards one answer at-a-time, while complex systems must reason about the whole meaning of the logic system. And, finally, the clausal form of rules, while being concise, hides logical quantifiers, making developers to keep them in mind to fully comprehend the logic equivalence of a rule.

Truly declarative languages include classical relational calculi (RC) (Silberschatz, Korth, & Sudarshan, 2010) (for relational databases), and Datalog (Green, Huang, Loo, & Zhou, 2013) (the *de facto* standard for deductive databases, but suffering also the last aforementioned drawback). All such drawbacks might be overcome with classical relational calculi if applied in a deductive database system supporting the relational data model. As another advantage, RC languages includes explicit use of both existential and universal quantifiers, which is not allowed in pure Datalog.

This data model was introduced as early as 1970 by E.F. Codd (Codd, 1970), and afterwards he developed in (Codd, 1972) two formal languages under such data model: a Relational Algebra (RA) and a Relational Calculus. This calculus is what we know nowadays as Tuple Relational Calculus (TRC) and led to the widely-used SQL language. Later, Lacroix and Pirotte (1977) introduced a variant of TRC: the domain relational calculus (DRC). Based on FOL, both TRC and DRC are truly declarative formal languages, which means that a query written in a calculus expresses *what* the set of data one wants to retrieve from the database, as opposed to *how*.

Such RC languages have been traditionally included in Computer Science Curricula (CSC) as, e.g., the Joint ACM/IEEE CSC 2013 (Curricula, 2013). But, despite their advantages (declarative, FOL formal languages, set-oriented), applying and teaching them can be better achieved if a comprehensive state-of-the-art system is available. Thus, in this work, we describe a recent addition to the deductive system DES (Sáenz-Pérez, Caballero, & García-Ruiz, 2011): both TRC and DRC in the same deductive database setting in which Datalog (the *de facto* query language for deductive databases), SQL and RA already coexist (Sáenz-Pérez, 2014). This system permits deductions with the inference engine it incorporates, deducing a set of data as the outcome of a query in any of such languages. As the inference engine implements a stratified fixpoint for deduc-

☆ Work partially supported by the Spanish MINECO project CAVI-ART (TIN2013-44742-C4-3-R) and Madrid regional project N-GREENS Software-CM (S2013/ICE-2731).

E-mail address: fernan@sip.ucm.es

https://doi.org/10.1016/j.eswa.2017.12.007
0957-4174/

tions, it can be effectively applied to the RC languages for solving recursive definitions and queries. Though the possibility of solving recursive RC queries has been introduced with fixpoint and while algorithms (Abiteboul, Vardi, & Vianu, 1995), to the best of our knowledge, we provide the first available system allowing recursion in RC languages. This opens a brand new ream of applications with RC languages, including those needing transitive closure, reachability (social networks), same generation problems and the like. In addition, it supports ODBC (Open DataBase Connectivity) connections, making it possible to seamlessly solve RC queries on data stored on external relational database management systems (RDBMS).

DES was originally intended to education, and adding these calculi also sets a new step towards this goal. This way, given for granted that the relational data structure model coincides with the deductive data model (Yazdanian, 1987) (atomic data are arranged in predicates which can be understood as relations, i.e., relational tables), all of these languages can retrieve data from the same relations. So, students are able to exercise their acquired knowledge about all these languages in a common setting, which can refer to the local in-memory deductive database and external relational databases.

Certainly, this is not the first system implementing these languages. There have been a few reported systems[1] such as the Relational Calculus Emulator (Bhandari, 2011) and more importantly WinRDBI (Dietrich, Eckert, & Piscator, 1997) (built after a first command-line processor implemented in Quintus Prolog (Dietrich, 1993)), which is described in the book (Dietrich, 2001) and used in many universities. The forthcoming Section 5 develops a more elaborated related work study, and a comparison of DES with current systems.

Despite the availability of these systems, several requirements conform the motivation for adding support for RC in DES. First, the original requirement was a feature request from an external university, in order to have available all formal languages in a single system. A second requirement is that, taking into account that systems implementing formal languages are almost completely targeted at students, more feedback from a practical system is needed in terms of syntax error reporting (compared to those available in other systems). To this end, we have included precise reports that focus on the error source with the aim that students read them and understand why their queries are not correct, in particular because safety (Ullman, 1988). Also, the graphical integrated development environment ACIDE (Sáenz-Pérez, 2007) has been configured for DES, helping students with syntax colouring to identify those possible errors when typing their queries. As a third requirement, we mention practicality, i.e., having all these languages in a single system capable of external connections to relational databases, with a comprehensive set of features in the languages and system settings, and a straightforward installation in several OS platforms (even with portable distributions and no root permissions).

So, in this paper we develop a system description of DES focusing in the inclusion of RC and the above-mentioned requirements. Section 2 introduces the syntax of TRC and DRC queries, including relation definitions in these languages, the notion of valid queries, propositional relations, and automatic type casting in a strong type system. Other textbook syntax is included as a reference, as well as some points about the implementation. Section 3 includes a description of the error system. Section 4 briefly describes applications of TRC and DRC for understanding SQL, SPARQL and QBE. As a sort of related work, Section 5 also includes a comparison with the two available systems incorporating RC languages. Finally, Section 6 concludes and highlights some points worth of future work.

[1] In contrast to the much more RA systems which have been developed and made available.

2. Relational calculi in DES

Relational calculus was proposed by E.F. Codd in (Codd, 1972) as a relational database sublanguage, together with a relational algebra with the same purpose. In that paper, he introduced what we know today as Tuple Relational Calculus (TRC) with a positional notation for relation attributes instead of the named notation more widely used nowadays. Whereas positional notation refers to relation attribute positions by a numeric index (as, e.g., `Employee.1` for the relation `employee(eID,...)`), named notation refers to relation attributes by their names (as `Employee.name`). Domain Relational Calculus (DRC) was proposed in (Lacroix & Pirotte, 1977) and includes domain variables as opposed to the tuple variables in TRC. Relational calculi express queries with logic constructs (conjunction, disjunction, negation, implication, and existential and universal quantifications), while RA includes operators including the Cartesian product, selection and projection. Both calculi are acknowledged as more declarative than their counterpart Relational Algebra (RA) because while the algebra requires to specify the order of the operations needed to compose the output data, the calculi do not (Levene & Loizou, 1999).

Next, before presenting TRC and DRC syntax, relations are introduced.

2.1. Relations

A relation schema $R(a_1, \ldots, a_n) : D_1 \times \ldots \times D_n$ is defined by its name R, its attributes with names a_i and corresponding domains (types) D_i. Extensional relations (i.e., tables) are defined in DES either with SQL Data Definition Language (DDL for creating, dropping, and modifying schemas) statements (`CREATE TABLE`, `ALTER TABLE`, ...) or with Datalog assertions (`:-type`, `:-pk`, ...). Available types for defining schemas are inherited from SQL and relation and attribute identifiers start either with lower-case or are delimited by double quotes (`"`, following ISO recommendation). Intensional relations (i.e., views) are specified with the assignment operator ($\leftarrow$) of formal relational languages, with the textual syntax $R := Q$, where R is the name of the relation and Q is a query. The attribute names can be specified as a tuple after R or automatically generated from the relations they are built. Argument types are inferred.

A relation instance $I_R \subseteq D_1 \times \ldots \times D_n$ is a set of actual tuples for a relation R. Instances of extensional relations are built in DES either with SQL Data Manipulation Language (DML for data updates, deletions and insertions) statements (`INSERT INTO`, ...) or with commands (`/assert`, ...) Instances of intensional relations are built via query solving.

2.2. TRC queries

Textual syntax of TRC statements (queries) in DES follows (Dietrich, 2001) (with a case-sensitive, named notation for user identifiers) but relaxing some conditions to ease the writing of queries. For instance, parentheses are not required unless they are really needed, but nonetheless they can be used sparingly to help reading. The basic syntax of a TRC query (alpha expression as known in (Codd, 1972)) is:

```
{ VarsAttsCtes | Formula }
```

where `VarsAttsCtes` is known as the target list: a comma-separated sequence of either tuple variables, or attributes, or constants: