Enterprise Modelling and Information Systems Architectures
Vol. 13, No. 2 (February 2018). DOI:10.18417/emisa.si.hcm.23

Special Issue on Conceptual Modelling in Honour of Heinrich C. Mayr

Understanding *Semantic Completeness* in Rule Frameworks for Modeling Cardinality Constraints

Faiz Currim[a], Sudha Ram[*,a]

[a] Department of Management Information Systems, Eller College of Management, University of Arizona, Tucson, USA

Abstract. *Modeling organizational rules during conceptual design provides a more accurate picture of the underlying domain and helps enforce data integrity. In a database development context, there are many advantages to explicitly representing rules during conceptual design. Early modeling ensures they are visible to designers and users, thus improving requirements and validation. The rules can then be semi-automatically translated into logical design code. One limitation to widespread adoption of such modeling is variance in standards and semantics of rules. We consider cardinality constraints—a useful and integral part of conceptual database design. Many papers discussing classification frameworks for cardinality exist. Completeness of such schemes has always been in question since well-defined criteria do not exist to evaluate them. We suggest a "reverse engineering" approach, i. e., one of defining conceptual modeling constraint completeness based on mappings from the relational model. We develop a correspondence from relational algebra operator combinations to existing semantic constraint types. In doing so, we also come up with a new category of set-level cardinality constraints not previously examined in literature. We believe our work demonstrates a unique approach to establishing conceptual framework completeness and enables standardization of rule semantics which in turn allows for semantics-based (as opposed to procedural-based) representation. On the implementation side, it supports developing automated mechanisms for translating constraints to improve developer productivity.*

Keywords. Database Design • Conceptual Database Modeling • Cardinality Constraints

1 Introduction and Motivation

Cardinality constraints have long been an integral part of conceptual database diagrams since the original entity-relationship (ER) model proposed by Chen (Chen 1976). Other conceptual modeling standards including Unified Modeling Language (UML) (OMG 2015), and Object-role modeling (ORM) (Halpin and Morgan 2010) also provide support for cardinality. As do pre-design models such as KCPM (Vöhringer and Mayr 2006). (The terminology may vary across models, e.g., UML or KCPM may term it as multiplicity.) A variety of papers have examined cardinality constraints in detail, and many frameworks and taxonomies have been proposed to comprehensively organize the types of cardinality constraints (Lenzerini and Santucci 1983; Thalheim 1992; Liddle et al. 1993; McAllister 1998; Ram and Khatri 2005). With any taxonomy, including one for cardinality, the question of semantic *expressiveness* or *completeness*[1] (Navathe et al. 1992) is pertinent. Establishing completeness is valuable from a standpoint of both theory and practice, and allows one to establish an exhaustive mapping into implemented database constraints. Though authors have

* Corresponding author.
E-mail. ram@eller.arizona.edu
We thank Nicholas Neidig, Alankar Kampoowale, Girish Mhatre, Anish Padiyara and Mark Vanderflugt for assistance with developing the CARD system prototype.

[1] We use the terms completeness, comprehensiveness and expressiveness interchangeably. Some writers prefer the term expressiveness since completeness often implies an absolute completeness which can be difficult to establish.

sought to address the issue of completeness of their frameworks, it has remained a difficult and open question. This is because it is hard to prove in advance that a scheme (e.g., conceptual model or a rule framework) is fully expressive. Options to demonstrate completeness include envisioning a large number of possible scenarios (including covering what similar models have used in the past) or extensive testing in the field. Each has its costs and drawbacks, can be time-consuming, and remain susceptible to not covering the needs of a new application. This can lead to long delays in adoption of a framework, since it may be impeded due to the perception of it being "yet another incremental version" that augments expressiveness by some small amount but is not complete.

Proving expressiveness in the context of rule frameworks is important, as it allows for a comprehensive set of constraint specifications and resultant translation into code. This in turn provides a much stronger case for incorporating rule frameworks into design tools. Having a visible rule repository is important for good managerial decisions (Von Halle 2001) and researchers and practitioners have recommended that user-specified business rules applicable to data should be documented in the database schema itself (ISO 1987; Simsion 2001). In data modeling, cardinality is one such class of rules that must be modeled.

In this work, we focus on cardinality rules.[2]

Our objective is to determine how completeness can be established for cardinality constraint frameworks. We discuss previous approaches to proposing comprehensiveness in the conceptual modeling domain, and define completeness in a novel way. Instead of viewing the transition from the conceptual to the logical design stage as a one-way street, we "reverse map" from relational algebra to conceptual modeling constraint kinds. We consider combinations of algebraic operations, and show that a complete framework must express constraints corresponding to all relevant operation arrangements. We also test the feasibility of such mapping with a proof-of-concept prototype system and present some findings and recommendations for DBMS support of these kinds of rules.

The rest of our paper is structured as follows. We begin by reviewing prior work in rule representation in conceptual and logical design in section 2. In section 3, we address the issue of completeness for semantic modeling and why our proposed approach is reasonable. Section 4, contains the discussion of the various relational algebra operators and how different combinations of them map to cardinality constraints (with a SQL mapping for each constraint kind). Thereafter, we discuss our evaluation and present our recommendations in section 5. Section 6 contains the conclusions and suggestions for future work.

2 Review of Related Work

We see support for representing a variety of rule types in conceptual database design. The original ER proposal allowed for representing identifying attributes[3] , cardinality, and implicitly—referential integrity through the specification of relationships. Extensions to the ER model have allowed designation of the mandatory vs. optional properties for relationship participation, as well as for attributes, i.e., whether they can contain null values (Figure 1). Newer modeling tools may further allow specification of additional constraints such as data types and domain ranges (see Figure 2) by providing an interface that bridges conceptual and logical design.

From a data management perspective, such rules function as integrity constraints on a database helping to ensure that the business policies and semantics of the application are incorporated into the database (Storey et al. 1996; Date 2000a; Simsion

[2] We use the terms *constraints* and *rules* synonymously though we recognize that there are multiple interpretations for "rule" in literature. We use *policy* to refer to the underlying business directive that guides the constraint.

[3] Some authors prefer the term "primary key"; however, the notion of primary keys comes from the relational model (logical design), and we adopt the usage of "identifying attributes" in context of conceptual design.

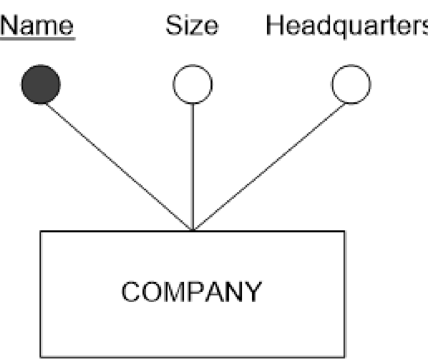


Figure 1: In this diagram, the attributes size and headquarters may contain null values (syntax from (Umanath and Scamell 2007))

2001). All well-known conceptual grammars[4] provide support for cardinality, as it is needed to determine the translation of a conceptual schema into the corresponding logical relations (for the purposes of our paper we assume the relational model is used during logical design). Similarly, cardinality is employed when mapping from a pre-design glossary to a conceptual schema (Mayr and Kop 2002). Cardinality rules are also useful for other database purposes including normalization (Navathe et al. 1992), schema integration (Ramesh and Ram 1997), query optimization (Thalheim 1996), and managing privacy for sensitive data (Sweeney 2002).

The importance of capturing and visibly representing business rules has been highlighted by a number of efforts (Date 2000b; Hay et al. 2000; Von Halle 2001; Ross 2005; OMG 2017).

However, inclusion of support for representing cardinality in modeling methodologies and tools has been somewhat limited. One of the issues with cardinality is the complexity in its semantics when generalized beyond a binary relationship case or when dealing with temporal or spatial data. Compounding this problem is the variations in intended semantics as used in different modeling grammars (Ferg 1991; Liddle et al. 1993). Over time, different approaches to address the problem have been proposed. UML, for example, allows the specification of complex cardinality constraints using the Object Constraint Language (OCL) (OCL 2014). Adapting an example from Warmer and Kleppe (J. Warmer and A. Kleppe 1999), consider the SubmitBids relationship schema in Figure 3. A requirement that "any supplier and project combination can appear in the SubmitBids relationship between [*h* .. *k*] times", where *h* and *k* are a user-specified integer bounds, can be specified using the OCL code in Figure 4.

A similar approach can be adopted to assert, for instance, that a student can take the same course up to *m* times (across semesters) or an employee may be assigned to work on the same project up to *n* times in a year. Being a procedural specification in OCL, this option is not without its own deficiencies. It does not scale well, and each time we see such a constraint—we must rewrite the logic for its implementation.

An alternative approach that exists in literature, is to specify a conceptual taxonomy of the various cardinality constraints, and then take advantage of the classification scheme to denote the constraint type and its parameters. Using a syntax formalized previously (Currim and Ram 2012), we can state the earlier participation[5] constraint on the SubmitBids relationship by:

```
CARD-R-PT (SubmitBids, (SUPPLIERS,
PROJECTS) ) IN [h:k]
```

Thus, conciseness of annotation is better achieved using a semantics-based or conceptual classification. This can lead to improved analyst productivity, and reduce the chance of errors while writing a program. An advantage of this method is that it can be mapped to the corresponding OCL

[4] We use the term *grammar* to refer to the formalism specifying the constructs and rules for conceptual design, e.g., ER Model. We use the term *schema* (or *script*) to refer to the abstract description of the real world developed using the constructs provided by the *grammar* (Wand and Weber 2002).

[5] Briefly explaining the syntax, in their scheme CARD-R-PT stands for a cardinality (CARD) constraint on an interaction relationship (R), restricting participation (PT). The parameters are the relationship involved, and the constrained entity classes.

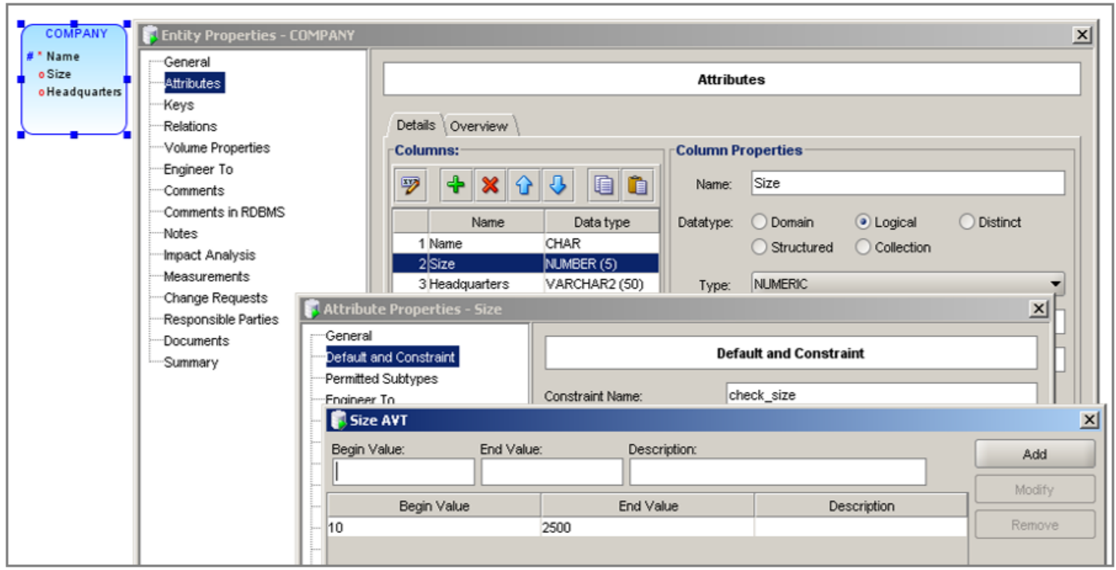


Figure 2: Using a conceptual design tool to specify the data type for the Size attribute, and a check constraint to be implemented upon conversion to relational table (i.e., logical-level) creation code

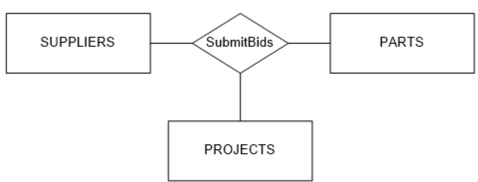


Figure 3: The Submit Bids relationship

specification or triggers (which we show later in this paper) in a straightforward manner. On the other hand, a limitation with this approach, as demonstrated by the augmented taxonomies published over the years (Lenzerini and Santucci 1983; Thalheim 1992; Liddle et al. 1993; McAllister 1998; Ram and Khatri 2005), is that establishing comprehensiveness in the organized types of cardinality constraints is difficult. Successive efforts have added support for new kinds of constraints. While this improves the body of knowledge, it is still desirable to have a sense of completeness so that modeling tools can incorporate the taxonomy and the related translation plans. Otherwise, an argument can be made that logical-level proposals that generate pseudo-code, while less efficient and insensitive to the underlying semantics, provide

```
sb-s: SubmitBids -> SUPPLIERS
sb-j: SubmitBids -> PROJECTS

SUPPLIERS -> forAll( s |
  PROJECTS -> forAll ( j |
    SubmitBids -> select (sb |
    sb.sb-s = s and sb.sb-j = j
    ) -> size >= h
    and
    SubmitBids -> select (sb |
    sb.sb-s = s and sb.sb-j = j
    ) -> size <= k
  )
)
```

Figure 4: Sample OCL code for specifying a participation constraint

the flexibility of incorporating new kinds of constraints. To enable wider adoption of taxonomic cardinality specification approaches for rule processing and service-oriented computing in a heterogeneous environment, we feel a standardized and *complete* framework for cardinality constraints is important.

3 Addressing Completeness

As mentioned earlier, addressing the question of rule framework completeness is valuable from both a research and business standpoint. In the context of semantic modeling and cardinality, while authors have sought to address the matter of completeness of their frameworks, it has remained an open issue. Completeness is difficult to measure because there is no easy way to establish a priori that a model has the necessary constructs to capture the semantics of every possible application. The task can be simplified somewhat by narrowing the scope of the taxonomy or model (which is one of the reasons, besides compactness of representation, that we see a proliferation of domain-specific conceptual grammars). Having reduced the target modeling space, one can test completeness by undertaking multiple case studies in a variety of organizations in the field. The greater the number of studies, the more confidence one has in the expressiveness of the taxonomy.

Since grammars like the Entity-Relationship model have already been extensively field-tested, some authors have adopted the tactic of measuring *relative completeness* where the new model is measured against existing grammars (Bajaj and Ram 2002). Thus, a conceptual model can be considered relatively complete if its constructs are at least as expressive as previously developed models. Applying the norm to constraint frameworks, we can say that a framework is relatively complete if the classified constraints incorporate those seen in existing constraint systems. Most work to date has implicitly adopted this approach while demonstrating that their proposed framework encompasses previous classificatory schemes (Liddle et al. 1993; McAllister 1998; Ram and Khatri 2005). An argument of insufficient confidence could, however, be made against this methodology since cardinality rules are not always thoroughly specified while developing a conceptual schema, and thus our faith in the completeness of existing classification mechanisms may not be strong.

To address this issue we asked ourselves, “What alternative benchmarks could be used for expressiveness?” There was no simple answer due to the absence of pre-existing criteria to define completeness. We knew that extensive organizational testing is not practical in a reasonable timeframe. The approach we chose instead was to adapt our test for completeness by taking advantage of existing work in relational query sub-languages (in our case, relational algebra), where completeness has already been well-defined.

Previous efforts on completeness for ER-based query languages (Atzeni and Chen 1981; Campbell et al. 1985) are not based on relational algebra or relational calculus, and instead have defined expressiveness based on constructs within the ER model. Our approach diverges from this, and we argue that our “reverse engineering” (working backwards from the logical design) strategy is acceptable for three reasons. Firstly, the underlying theoretical model for both the entity relationship and the relational models is the same, i.e., set theory. The constructs for capturing and storing data can be visualized as a derivation of sets in both models. This, in part, accounts for the well-defined and straightforward mapping between the two models. Secondly, since relational language completeness has been well-tested (and can thus be construed as a sound measure), we may as well take advantage of this knowledge while formulating a measure of completeness for constraints. Finally, we feel that since the implementation of conceptual model cardinality constraints will typically be done in a relational database, it would be reasonable to think that a classification scheme is complete only if every cardinality constraint type that can be implemented in a relational algebra is also available in a conceptual model cardinality framework.

4 Establishing Completeness Using Relational Algebra

To begin, we intuitively establish the correspondence between constraints and queries. Let us take the example of a constraint: *An employee can work for between 1 and 25 projects* specified on the work_on relationship (see Figure 5). In the relational model, this relationship would map to a table work_on and an SQL query would be executed to perform a count of projects for the employee in the modified tuple. Depending on the results, the system could determine if the constraint were satisfied or violated.

Correspondingly, we argue that the evaluation of every cardinality constraint can be mapped to an SQL query (or relational algebra expression) to be checked upon a database operation (insert, delete or update). Since the query to evaluate a cardinality constraint is but one kind of query, the set of all possible cardinality constraint queries is a strict subset of all possible queries. Thus, a language that is constraint-complete will be a subset of a language that is relationally complete. Next, we discuss the operators required for relational completeness, and the relevant subset needed for checking cardinality constraints.

For a language to be relationally complete, it must be capable of the following relational operations (Codd 1972): projection (π), selection (σ), Cartesian product ($\times$), union ($\cup$), and set difference ($-$). Further, we consider a common extension, that of supporting aggregation and grouping ($\Im$) and including functions (like counts) on attributes to be performed in a projection. As might be expected, we do not need the full functionality of all the relational operators π, σ, $-$, $\Im$, $\times$, $\cup$ for checking cardinality constraints. We elaborate below.

The projection π operator is unique in that it filters attributes rather than tuples (or in other words: extracts properties rather than entities). In a general query language, any combination of attributes and expressions may be desired in the output and therefore the use of the π operator is diverse. However, for checking a cardinality constraint—the only kind of query we are interested in is a count. We denote the count based on a projection with the symbol π^{c}. Two different kinds of counting may be performed. The simplest is to count entire tuples (SELECT COUNT (*) in SQL), which we denote by π^{c^*}. Alternatively, a count of distinct attribute values may be carried out (SELECT COUNT (DISTINCT ⟨attribute⟩)), which we denote using π^{cA}.

The operators of Union and Set Difference $\{\cup, -\}$ serve to either augment or reduce the membership of the set under consideration (leading to the creation of a new set). They are similar in that they change the set membership, but not how elements are counted within a defined set and therefore do not change the nature of the cardinality constraint being considered. Thus, they are not as important from the perspective of a cardinality framework.

The Cartesian product $\{\times\}$ is usually performed in conjunction with the selection operator σ to create a “join”. Conceptual model constraints often implicitly assume the use of joins. For example, the constraint: *An employee with the designation of “Senior Consultant” can work for between 1 and 10 projects*, defined on the work_on relationship (Figure 5), contains a predicate on the participating EMPLOYEES entity class, and hence implicitly uses a join (when considered in the relational model). The constraint frameworks we have encountered thus far only consider joins within a relationship’s participating entity classes, rather than joins among classes that may span a chain of relationships (i.e., they would not consider a constraint defined between employees and clients in Figure 6). To address this, we propose the notion of *virtual relationships*, discussed in more detail elsewhere (Currim 2004); which naturally maps to the definition of a logical view. A point to note is that the *nature of counting* that is performed *does not change*, merely the scope. Thus, we can continue our analysis excluding such joins without loss of generality.

Finally, we consider the two operators that when applied to a set allow us to count specific members or aspects of that set. Both selection

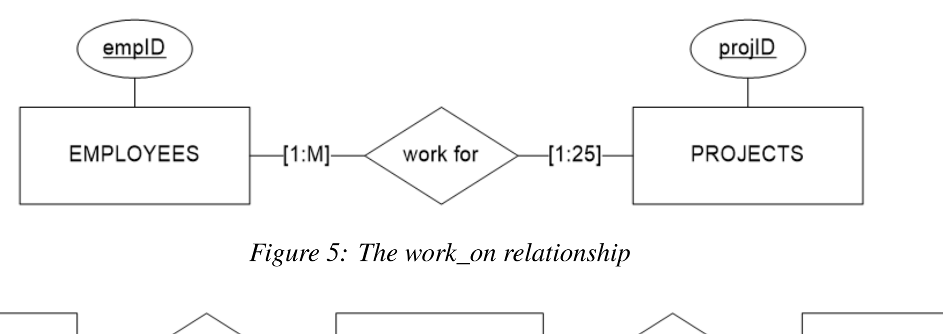


Figure 5: The work_on relationship

EMPLOYEES [1:M] work on [0:M] PROJECTS [0:M] request [1:1] CLIENTS

Figure 6: Employees, Projects and Clients

and grouping allow us to count specific aspects of the set (or specific members). This determines what is counted, and hence the kind of cardinality constraint being examined. We examine both these operators in more detail.

Selection (σ) may be performed on any attribute combination within one or more relations. We decompose this further into: σ performed on an identifying attribute of a participating entity (or entity combination), denoted by σ^{ID}, and a σ performed on a non-identifying attribute σ^{A}. We separate the two because there is a direct correspondence between an identifier and an entity instance, which in turn allows us to ask the question "how many" (i.e., count) as applicable to a given entity (e.g., employee). While there has been some deliberation in literature about the distinction between entities and attributes, we do not get into that discussion and instead refer the reader to previous work (Weber 1996; Kop and Mayr 1998), while assuming that the distinction is beneficial. Related to the use of σ^{ID}, we know from the query languages that a σ need not contain only a single identifier value, and could instead be evaluated over a set of identifiers using either multiple `OR` clauses, (in this case we can combine the set of identifiers using an `IN` operator as well). We denote a set of identifiers by: $\sigma^{\text{ID-Set}}$. This leads to the classification of a conceptual model constraint category not previously discussed in literature, that of *set-level constraints* (e.g., *There should be no more than 5 projects associated across the set of employees: 'E004', 'E005', 'E007'*).

Returning to σ^{A}, when the attribute A is from a participating entity class, the σ^{A} serves to establish a predicate on the relevant entity class. This does not impact the nature of the constraint, just the scope of the set membership on which it is defined. For example, consider constraints:
C1a: *An employee can work for between 1 and 25 projects;* and C1b: *An employee with the designation of "Senior Consultant" can work for between 5 and 10 projects* (this version also considers a *single employee*, but has an additional σ condition to only consider an employee if they are a senior consultant).

The fundamental form of both these constraints is the same in that they capture the number of members of a "counted" entity class (projects) that co-occur with each member of a "fixed" class (employees). The only difference between them is a predicate that restricts the membership of the "fixed" set (or the "counted" set). This principle holds whether we are considering instance-level or set-level constraints. As an example, consider the following constraints:
C2a: *There should be no more than 50 projects associated with the set of all employees;* and C2b: *There should be no more than 5 projects associated with the set of all employees with a security clearance of "alpha"* (this version also considers a *set of employees*, but has an additional

σ condition to only consider those employees with a security clearance of “alpha”).

Both these constraints are set-level constraints where the “fixed” class is employees, and the cardinality limits the association with projects (the “counted” class). To summarize, using predicates provides flexibility in expressing a constraint type, but does not change the nature of what is counted.

The case may be made for separate consideration of attributes in σ^{A} that are mutual properties of entities in a relationship, e.g., counting employees who received a particular level of feedback or rating for their work on the project. We adopt a philosophy previously presented in literature suggesting that relationships should not have attributes of their own, but these must be either modeled via a separate relationship or attaching an additional associated (usually, weak) entity class (Wand et al. 1999). The function of these attributes in constraints consequently reduces to that of predicates. Even if we do not adopt this view, the only difference is that we end up with a generalized version of constraint types that allow for counting of not only entity instances but also attribute values within a relationship. Either way, the target relational constraint can be captured using conceptual model constraints.

The grouping operation $\mathfrak{I}$ is a convenient extension of the σ applied to specific entity instances. Instead of checking the count for a single instance, it does so for each distinguishable instance in a relationship. For example, we could perform a count of projects for a single employee by specifying an employeeID with the σ operator. Using $\mathfrak{I}$ allows us to conveniently perform a similar count for each employee without having to manually enumerate each employee’s identifier. When a σ is used in conjunction with a $\mathfrak{I}$, it (the σ) serves the role of either restricting the scope of the $\mathfrak{I}$ (if the attributes the σ and $\mathfrak{I}$ are applied on are identical), or that of predicate filtering. As is the case for σ, we consider $\mathfrak{I}$ on identifying attributes without loss of expressiveness.

We present our analysis of the semantics of cardinality in the next two sections. The basic approach followed is to examine combinations of projection, selection and grouping and their correspondence to various cardinality constraints. For consistency, we assume the underlying conceptual schema is developed using the ER model (Appendix A summarizes the syntax used), and follow cardinality terminology developed by Liddle (Liddle et al. 1993) and extended by us (Currim and Ram 2012). These papers also contain a resolution among constraint naming conventions used in a variety of semantic models and previous frameworks. To formally clarify the semantics of each constraint type, we use first order logic and integrate it with a sample annotation syntax (recapped in Appendix B using BNF). A (simplified) corporate schema[6] used to illustrate the constraint semantics follows.

4.1 CONSTRAINTS APPLICABLE TO ENTITY CLASSES

A **class** constraint restricts the number of members in the entity class. An **attribute** constraint restricts the cardinality of the number of values an entity instance can have (for an attribute), e.g., we may wish to restrict how many cities a project can span. A **domain** cardinality constraint (encountered and formally defined by us during the process of establishing completeness relative to relational algebra) restricts the number of possible values an attribute can take on for the set of entity instances. The domain cardinality does not restrict the number of cities for a *single* project entity, rather specify a restriction *across all* (or some subset of) projects.

The cardinality of a finite set E (e.g., entity class) is written as $|E|$. Typically, the constraint is specified with a lower and upper bound. We represent the range of values between the bounds by the set *Card*. Since E is finite, we say $|E| \in Card$, where $Card \subseteq \mathbb{N} \cup \{\mathtt{M}\}$; $\mathbb{N}$ is the set of natural numbers and the symbol $\mathtt{M}$ denotes “many” (unrestricted upper bound). For convenience, in

[6] To preserve the ternary semantics of cardinality constraints (as these are easier to read and interpret), we don’t model the change in cardinality due to the introduction of a weak association class and the consequent change in degree of the relationship to quaternary.

the syntactical version, we specify the lower and upper limits of the constraint as ⟨min⟩ and ⟨max⟩, where $\langle min \rangle \subseteq \mathbb{N}$, and $\langle max \rangle \subseteq \mathbb{N} \cup \{\mathtt{M}\}$. We use *A* for attributes, and *P* to represent a predicate. $P(e(A_{km}))$ signifies the mth predicate defined on attribute A_k. For simplicity, we only describe the use of predicates for entity class cardinality and do not repeat the syntax for the constraint bounds (i.e., "`IN` [⟨min⟩:⟨max⟩]") for later constraints. For an entity class *E*, we use $\pi^{A}(E)$ to denote the projection of the values of attribute A across all members of *E*, while $\pi^{A}(e)$ represents the projection for a specific entity *e*, where $e \in E$.

In Table 2 we consider combinations of relational algebra operators as they apply to class cardinality constraints. In Table 3, we use π^{cA} operations instead of π^{c*}, which leads to attribute and domain cardinality constraints. In the first column we show the relational algebra expression, followed by the semantics of the specified constraint (natural language). The final column has the equivalent SQL query.

In the next sub-section, we discuss constraints applicable to relationships.

4.2 CONSTRAINTS APPLICABLE TO RELATIONSHIPS

An *interaction relationship* relates members of one entity class to members of one or more entity classes. Interaction relationship constraints are classified into participation, set-participation, projection, co-occurrence, set-co-occurrence, appearance, set-appearance, appearance-across-R, and set-appearance-across-R constraints (the latter four types being applicable for unary relationships). We begin by briefly describing each type of constraint and illustrating its semantics using the relationship assign from Figure 7. Thereafter, each is discussed in detail. To formally clarify the semantics of each constraint type, we use first order logic.

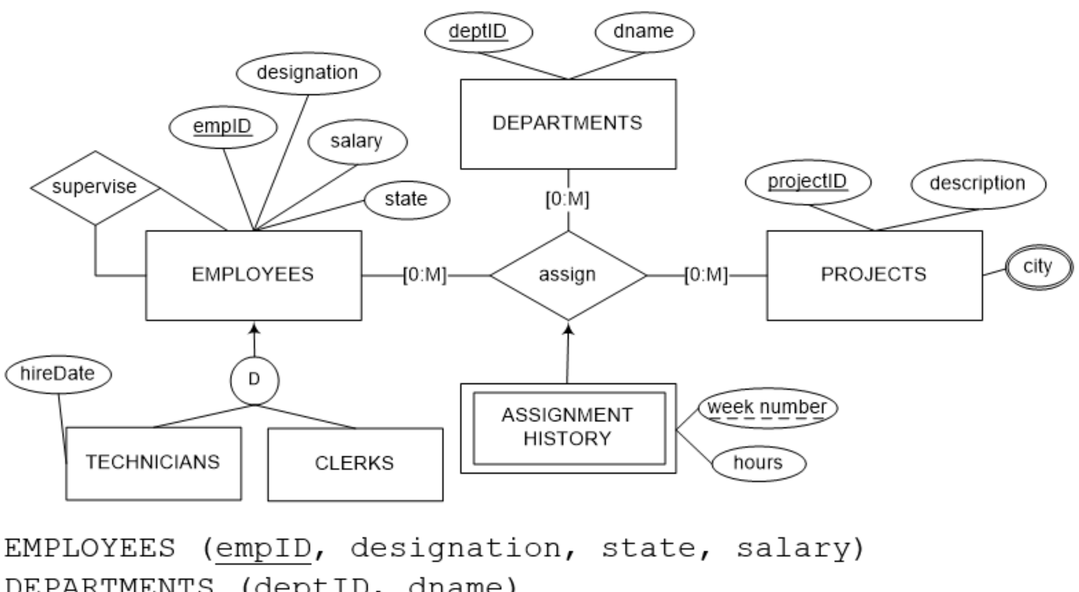
Participation constraints look at a relationship (e.g., employees being *assigned* to projects by departments), and ask the question, "How many times can an entity (e.g., an employee) participate in the relationship?" **Set-Participation** constraints (newly introduced by us) look at a (sub)set of entities (e.g., employees belonging to the states of California, Arizona and New Mexico) and ask, "How many times can the set of entities considered together participate in the relationship?" The generalized version of both these rule types consider not just a single entity class, but also entity combinations, for example "How many times can a given combination of employee and department be assigned in the assign relationship?"

For the formal definition, we introduce *entity from relationship* projection. Assume a relationship *R* (e.g., the assign relationship) is formed by the entity classes $E_1, \ldots, E_i$ and each element $r \in R$ (e.g., a specific assignment instance). Then we define $\pi^{E_i}(r)$ as the projection of the entity from class E_i belonging to the relationship instance *r*, while $\pi^{E_i}(R)$ projects all members from E_i, present within the relationship *R*. For the set-participation constraints, to define a subset C_i of an entity class E_i, we use $C_i \subseteq E_i$.

Projection constraints look at the relationship and restrict how many distinct entity instances can occur across the set of relationship instances. Thus, "How many different projects can there be in all (in the assign relationship)?" is an example of the projection constraint. The generalized version of this constraint examines entity combinations from multiple entity classes.

Co-occurrence constraints consider an entity already known to be participating in a relationship, and ask how many members of another entity class can co-occur with it. Thus, for example, one could ask, "Given a project that exists in the assign relationship, how many distinct departments can co-occur with it?" The *Set-Co-occurrence* constraint (newly introduced by us) looks at a set of entities known to be participating in a relationship and asks how many members of another entity class can co-occur with it. For example, "For projects classified as high-security, how many different meetings co-occur with them?" The generalized version of this constraint considers entity combinations.

So far, we have only examined cardinality constraints defined over instances of a relationship.




```
EMPLOYEES (empID, designation, state, salary)
DEPARTMENTS (deptID, dname)
MANAGERS (empID, hireDate)
PROJECTS (projectID, description)
PROJECT_CITIES (projectID, city)
SUPERVISE (empID, supervisorID)
ASSIGN (projectID, empID, deptID, weekNumber, hours)
```

Figure 7: A simplified corporate schema (ER and Relational versions)

Table 1: Semantics and sample syntax for Class and Attribute cardinality constraints

Class Cardinality	`CARD-C` (E) `IN` [⟨`min`⟩:⟨`max`⟩] *is defined as:* $\lvert\{ e : e \in E \}\rvert \in Card$
(with predicates)	`CARD-C` (E [⟨`predicate conditions`⟩]) $\lvert\{ e : e \in E \wedge P(e(A_{11}), e(A_{12}), \ldots, e(A_{k1}), \ldots, e(A_{km})) \}\rvert \in Card$
Attribute Cardinality	`CARD-A` (E, A) $\forall e \in E, \lvert \{ \pi^{A}(e) \} \rvert \in Card$
Domain Cardinality	`CARD-D` (E, A) $\lvert \{ \pi^{A}(E) \} \rvert \in Card$

Including the notion of counting values across attributes rather than tuples (i.e., the participating entity classes, rather than the relationship instances) leads us to the examination of forms of the *appearance constraint*. These are applicable for unary relationships. For example, we may ask, "*How many roles supervisee, supervisor can a single employee play within a single relationship instance of the* supervise *relationship?*" Thus, if an employee can supervise herself, then she can play two roles. This is an example of the **appearance** constraint, which restricts the number of roles in which a given member *e* of an entity class *E* can appear in any instance *r*, $r \in R$. It applies to an interaction relationship *R* in which the same underlying entity class *E* participates in *R* in different roles $L_1, \ldots L_k$. The **Set-Appearance** constraint restricts the number of roles a set of entities can play in any single entity instance. An **Appearance Across-R** constraint restricts the number of roles in which a given member of *E* can appear across all instances of *R* (or some subset *R*', $R' \subseteq R$). A

Table 2: Semantic Analysis of Combinations of SQL Operations for Entity Classes

Algebraic Operator	Semantics for an Entity Class	SQL Equivalent
π^{c^*} i.e., applied on the entity (tuple)	*Class cardinality* constraint (number of members in an entity class)	`SELECT count(*) FROM employees;`
π^{c^*}, σ	*Class cardinality* with predicate	`SELECT count(*) FROM employees WHERE state='CA';`
π^{c^*}, $\Im$	*Class cardinality* of an attribute-defined subclass/subset (while the equivalent cardinality may be computed by using state as a discriminator to define subclasses—we feel that it is somewhat artificial to require creation of subclasses simply to enforce cardinality). *Note:* it is not meaningful to use the identifier as the grouping attribute—since that will always yield one group per entity (i.e., any further count will always equal to 1)	`SELECT count(*) FROM employees GROUP BY state;`
π^{c}, $\Im$, σ	*Class cardinality* of a subset/subclass (based on predicate)	`SELECT count(*) FROM employees WHERE salary > n GROUP BY state;`
π^{c}, -	For completeness, we illustrate the effect of combining queries using the `MINUS` operator. As can be seen, it is not meaningful to perform such counts since it will return the count of the first query unless the cardinalities of the two sets in question are identical, whereupon it will return no results. Instead, the set difference should be performed first, and a single count taken on the resulting set (which does not change the nature of what is counted).	`SELECT count(*) FROM employees MINUS SELECT count(*) FROM managers;`

Set-Appearance Across-R constraint restricts the number of roles in which a given set C, $C \subseteq E$, of entities can appear across R (or some subset R', $R' \subseteq R$). We discuss the formal semantics and syntax in Table 7. For convenience, we define role projection operations. Role projection $\pi^{L}(r, e)$ is defined as: $R \times E \rightarrow \boldsymbol{P}(L)$, where $\boldsymbol{P}(L)$ is the power-set of L. It takes as input a relationship instance r, an entity instance e, and returns the set of roles that entity instance plays in the instance r. We generalize this to allow for $\pi^{L}(r, C)$, where $C \subseteq E$, and define it as: $R \times \boldsymbol{P}(C) \rightarrow \boldsymbol{P}(L)$. Similarly, we define operations to allow for projection of roles across instances of R, both: $\pi^{L}(R, e)$ and $\pi^{L}(R, C)$. Doing so allows us to keep the definition of the constraints compact.

Now that we have described the formal semantics for the various relationship constraint types, we consider combinations of relational algebra operators and SQL as they apply to the relevant cardinality constraints.

For appearance constraints, the query is non-

Table 3: Semantic Analysis of Combinations of SQL Operations for Attributes and Domains

Operator	**Semantics for an Entity Class**	**SQL Equivalent**
π^{cA} i.e., applied on a single attribute	*Domain cardinality. Note*: it is not useful to count the identifying attribute—since that will always be equal to the cardinality of the class itself	`SELECT count(distinct designation) FROM employees;`
π^{cA}, σ	*Domain cardinality* with a predicate (if non-identifying attribute in the selection σ)	`SELECT count(distinct designation) FROM employees WHERE state='CA';`
π^{cA}, σ^{ID}	*Attribute cardinality* for a specific entity instance (if part of the identifying attribute is included in the count). This is only meaningful for a multi-valued attribute (which in relational terms would be translated into a separate table)	`SELECT count(distinct city) FROM project_cities WHERE projectID='J15';`
π^{cA}, $\mathfrak{I}$	*Domain cardinality* within each defined subset	`SELECT count(distinct designation) FROM employees GROUP BY state;`
π^{cA}, $\mathfrak{I}$, σ	*Domain cardinality* within each defined subset (with a predicate)	`SELECT count(distinct designation) FROM employees WHERE salary > n GROUP BY state;`
π^{cA}, -	As discussed for the π^{c*} case, it is not meaningful to perform counts in this manner.	`SELECT count(distinct state) FROM employees MINUS SELECT count(distinct state) FROM managers;`

Table 4: Semantics and syntax for Participation (instance and set-level) constraints

Participation (generalized)	`CARD-R-PT` ($R, E_1, \ldots, E_i$) $\forall$ $(e_1 \in E_1, \ldots, e_i \in E_i)$, $\lvert\{r : r \in R \wedge \pi^{E_1}(r) = e_1 \wedge \ldots \wedge \pi^{E_i}(r) = e_i\}\rvert \in Card$
Set Participation	`CARD-R-PT-SET` $(R, E_1, \ldots, E_i)$ $\lvert\{r : r \in R \wedge \pi^{E_1}(r) \in C_1 \wedge \ldots \wedge \pi^{E_i}(r) \in C_i\}\rvert \in Card$

Table 5: Semantics and sample syntax for Projection constraints

Projection	`CARD-R-PJ` $(R, E_1, \ldots, E_i)$ $\lvert\{r : r \in \pi^{E_1,\ldots,E_i}(R)\}\rvert \in Card$

trivial since SQL does not have an in-built function to project roles. Unlike the extended projection operators defined in Table 7 (which allow for working with power sets), we further augment the

Table 6: Semantics and sample syntax for Co-occurrence (instance and set-level) constraints

Co-occurrence	`CARD-R-CO` $(R, (E_1, \ldots, E_i),(E_{i+1}, \ldots, E_j))\ \forall s \in \pi^{E_1,\ldots,E_i}(R), \lvert\{t : t\ \pi^{E_{i+1},\ldots,E_j}(R) \wedge (s \circ t \in R)\}\rvert \in Card$
Set Co-occurrence	`CARD-R-CO-SET` $(R, (E_1, \ldots, E_i),(E_{i+1}, \ldots, E_j))\ \lvert\{t : t \in \pi^{E_{i+1},\ldots,E_j} \wedge s \in \pi^{E_1,\ldots,E_i} \wedge (s \circ t \in R) \wedge \pi^{E_1}(s) \in C_1 \wedge \ldots. \wedge \pi^{E_i}(r) \in C_i\ \rvert \in Card$

Table 7: Semantics and sample syntax for various kinds of Appearance constraints

Appearance	`CARD-R-AP` $(R, E, L_1, \ldots, L_i)\ \forall e \in E, \lvert\{ l : l \in E\ \pi^{L}(r, e)\}\rvert \in Card$
Set Appearance Across-R	`CARD-R-AP-SET` $(R, E, L_1, \ldots, L_i)\lvert\{ l : l \in \pi^{L}(r, C) \wedge C \subseteq E\}\rvert \in Card$
Appearance Across-R	`CARD-R-APAR` $(R, E, L_1, \ldots, L_i)\ \forall e \in E, \lvert\{ l : l \in E\ \pi^{L}(R, e)\}\rvert \in Card$
Set Appearance Across-R	`CARD-R-APAR-SET` $(R, E, L_1, \ldots, L_i)\lvert\{ l : l \in \pi^{L}(R, C) \wedge C \subseteq E\}\rvert \in Card$

Table 8: Using only π

Algebraic Operator	**Semantics for an Entity Class**	**SQL Equivalent**
π^{c*}	*Projection cardinality*, i.e., number of association instances in the relationship.	`SELECT count(empID) + count(supervisorID) FROM assign;`
π^{cA}	*Projection cardinality* restricted to a participating entity class.	`SELECT count(distinct empID) FROM assign;`
$\pi^{cA,B}$	*Projection cardinality* restricted to a combination of entities from participating entity classes. We don't consider concatenation (or correspondingly: adding multiple attributes in the $\mathfrak{I}$ clause) in further examples to preserve simplicity, but note that it allows for combinations of entities from participating classes to be considered rather than just from one class.	`SELECT count(distinct concat(empID,deptID) ) FROM assign;`

functions in the presence of multiple roles (see Table 11 for details). This also demonstrates the advantage of using a semantics-based approach, rather than re-specifying the programming logic for each instance of the constraint. For simplicity, we assume a role-projection function exists for relational algebra and demonstrate one set of feasible solutions assuming two possible roles (based on the supervise relationship).

In this section we discussed the equivalence of conceptual modeling constraints and combinations of relational algebra operations. In doing so, we introduce new constraints at the set level. A similar exercise can be carried out for other modeling constructs (e.g., superclasses and subclasses). However, we feel the relational algebra mappings for entity classes, attributes and interaction relationships sufficiently demon-

Table 9: Using π^{c^*}, σ and $\Im$

Operator	**Semantics for an ER Relationship**	**SQL Equivalent**
π^{c^*}, σ^{ID}	*Participation cardinality*, i.e., number times an entity (or entity combination) participates in a relationship.	`SELECT count(*) FROM assign WHERE projectID='P01';`
π^{c^*}, $\Im^{ID}$	*Participation cardinality* (for every projectID in the relationship).	`SELECT count(*) FROM assign GROUP BY projectID;`
π^{c^*}, σ^{A}	*Projection cardinality*, i.e., number of association instances in the relationship matching the property specified in the σ.	`SELECT count(*) FROM assign WHERE hours = n;`
π^{c^*}, $\Im^{ID}$, σ^{A} or π^{c^*}, $\Im^{ID\text{-}x}$, $\sigma^{ID\text{-}y}$	*Participation cardinality*. The combination of σ and $\Im$ (performed on the same attribute) simply restricts which tuples of the entire set are split into groups. A $\Im$ performed on a different attribute than the σ leads to a participation cardinality check with the σ performing the role of predicate filtering. It remains a form of participation cardinality. We use $\Im^{ID\text{-}x}$, $\sigma^{ID\text{-}y}$ to indicate that the grouping and selection are on different identifying attributes (i.e., different entity classes). The $\sigma^{ID\text{-}y}$ condition may involve a single value or a set of values. The accompanying SQL example uses a set of values. A combination of σ^{A} with $\sigma^{ID\text{-}y}$ may also be done. A similar pattern is observed for co-occurrence constraints in Table 10 (discussion not repeated in the interests of brevity).	`SELECT count(*) FROM assign WHERE hours > n GROUP BY projectID;` `SELECT count(*) FROM assign WHERE empID IN ('E04', 'E05', 'E07') GROUP BY projectID;`
π^{c^*}, $\sigma^{ID\text{-}Set}$	*Set-Participation cardinality*, i.e., how often do a set of entities considered together participate in a relationship.	`SELECT count(*) FROM assign WHERE projectID IN ('P01','P02','P03');`

strates the important aspects of our approach.

5 Evaluation and Testing

To complement our approach for showing completeness of a cardinality taxonomy, we decided to develop a prototype system to serve as a proof-of-concept for the framework. To properly automate the development of the constraint translation module, we realized that the system needed to be aware of the relational schema (to reference column and table names in the SQL and triggers, for example). While our over-arching purpose was to evaluate whether the SQL mapping logic from our conceptual constraint specifications was correct, we felt the experience would additionally inform us of any database-implementation issues that could arise for the translated constraints. This motivated our development of the CARD (Constraint Automated Representation for DBMSs) system (Figure 8).

Table 10: Using π^{cA}, σ and $\Im$

Operator	**Semantics for an ER Relationship**	**SQL Equivalent**
π^{cA}, σ^{ID}	*Co-occurrence cardinality*, i.e., for an entity defined in the σ clause, how many distinct entity instances co-occur with it (π clause)?	`SELECT count(distinct empID) FROM assign WHERE projectID='P01';`
π^{cA}, $\Im^{ID}$	*Co-occurrence cardinality* for each entity defined by the grouping attribute(s).	`SELECT count(distinct empID) FROM assign GROUP BY projectID;`
π^{cA}, σ^{A}	*Projection cardinality*, restricted to a subset of association entities (matching the property specified in the σ) from the participating entity class (specified by the π^{cA}).	`SELECT count(distinct empID) FROM assign WHERE hours > n;`
π^{cA}, $\Im^{ID}$, σ^{A} or π^{c*}, $\Im^{ID\text{-}x}$, $\sigma^{ID\text{-}y}$	*Co-occurrence cardinality*. When used in combination, the grouping attribute(s) end up being the fixed aspect of the co-occurrence cardinality (it does not matter that a subset of entities is specified by the σ, since these are broken up for individual consideration by the $\Im$). The projection (counted) attribute plays its usual role, while the attribute in the σ clause works to filter the set of rows under consideration similar to a predicate.	`SELECT count (distinct empID) FROM assign WHERE projectID IN ('P01', 'P02', 'P03') GROUP BY deptID, projectID;`
π^{cA}, σ^{A} or π^{cA}, $\sigma^{ID\text{-}Set}$	*Set co-occurrence cardinality*, i.e., for a given set of entities (fixed by σ) how many instances of another entity class are associated with them (counted in π).	`SELECT count(distinct empID) FROM assign WHERE projectID IN ('P01', P02');`

Users interact with the system and choose one of the options to input a schema and associated constraints. The system then uses knowledge of standard ER-to-relational conversion logic to develop the SQL (DDL statements) for table creation. In addition, it takes advantage of the knowledge of the schema and the cardinality-to-SQL mapping we described in our paper, to generate the associated triggers. Currently, freeware and commercial tools exist that can do the first part (i.e., take a conceptual schema developed using ER modeling or UML and convert it into relations with a limited number of structural constraints). The latter part is new, and developed as an extension by us to demonstrate the practical feasibility of the translation based on our completeness discussion. A system like CARD can serve the software development lifecycle (SDLC) by generating relational triggers to manage the advanced constraints, which has the two-fold advantage of improving both database integrity and programmer productivity.

The prototype (currently implementing a subset of the cardinality constraints) has been developed in Java, and is available over the Internet (Currim et al. 2010). For our prototype we picked Oracle as the target DBMS, since it is widely-used. While the SQL table creation code is designed to be ANSI compliant and work across platforms, the constraint triggers are generated in PL/SQL and are Oracle specific (however, the core logic can

Table 11: Limiting roles (for Appearance constraint variants)

Operator	Semantics for an ER Relationship	SQL Equivalent
$\pi^{L(r,e)}$	*Appearance cardinality*, i.e., restricts how many roles a given entity can play in a single relationship instance. *Note*: while we use the Oracle specific DUAL system table, this can be translated into another platform like SQL Server either by removing the reference to DUAL (or for DB2: by using the SYSIBM.SYSDUMMY1 instead) or creating a schema specific table that simply contained a single row/column to use instead of dual.	`SELECT COUNT(*) FROM (SELECT 'One Role' FROM dual WHERE EXISTS (SELECT * FROM supervise WHERE (empid='E01' OR superviseID='E01') ) UNION SELECT 'Two Roles' FROM dual WHERE EXISTS ( SELECT * FROM supervise WHERE empid=superviseID AND empid='E01') );`
$\pi^{L(R,e)}$	*Appearance Across-R cardinality* (new). This constraint restricts the number of roles a single entity can play across all relationship instances. In this case, a UNION ALL (bag semantics) is required to prevent loss of information when the same entity (identifier) is selected from two different roles.	`SELECT COUNT(*) FROM (SELECT distinct empID FROM supervise WHERE empid = 'E01' UNION ALL SELECT distinct supervisorID FROM supervise WHERE supervisorID = 'E01');`
$\pi^{L(r,C)}$	*Set Appearance cardinality* (new). This constraint restricts the number of roles a set of entities can play in a single relationship instance.	`SELECT COUNT(*) FROM (SELECT 'One Role' FROM dual WHERE EXISTS (SELECT * FROM supervise WHERE (empid IN ('E01', 'E02') OR superviseID IN('E01', 'E02')) ) UNION ALL SELECT 'Two Roles' FROM dual WHERE EXISTS (SELECT * FROM supervise WHERE (empid IN ('E01', 'E02') AND superviseID IN('E01', 'E02')) );`
$\pi^{L(R,C)}$	*Set Appearance Across-R cardinality* (new). This is the set equivalent of the appearance across-R constraint and restricts how many roles a set of entities can appear in, across all relationship instances.	`SELECT COUNT(*) FROM (SELECT 'Role Employee' FROM dual WHERE EXISTS (SELECT * FROM supervise WHERE empid IN ('E01', 'E02') ) UNION ALL SELECT 'Role Supervisor' FROM dual WHERE EXISTS (SELECT * FROM supervise WHERE superviseID IN ('E01', 'E02') ));`

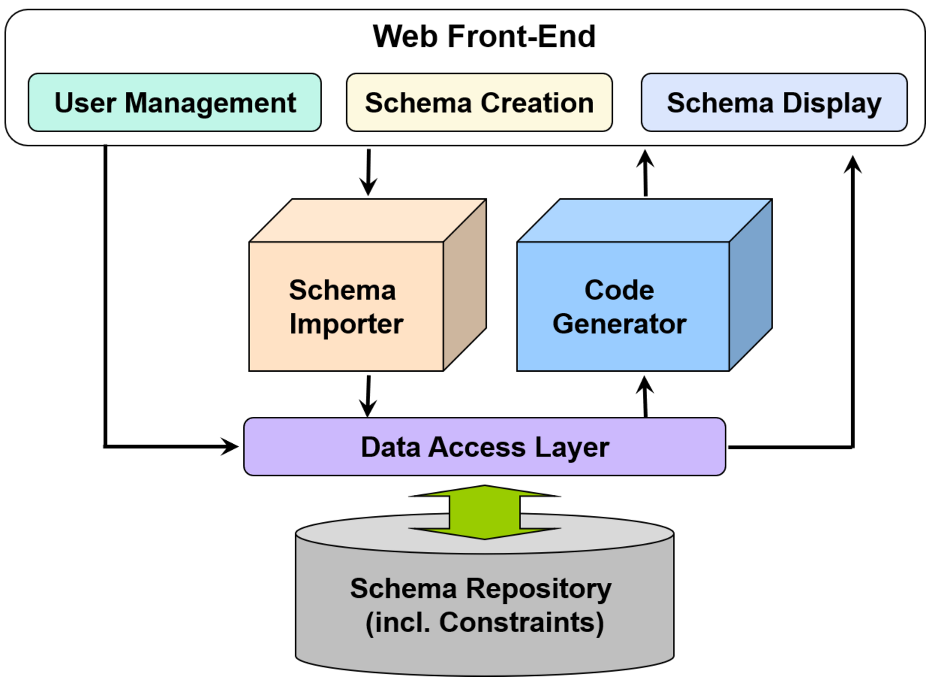


Figure 8: Architecture of the CARD system

Table 12: Implication of constraint parameters for triggering events

Constraint Property	Trigger Fired On
Minimum cardinality specification of > 0	DELETE
Maximum cardinality of < M(any)	INSERT
Predicates on Entity Class involved in Constraint	UPDATE

be modified in a straightforward manner for other platforms). The current interface supports creating schemas directly via a guided specification interface, and importing schemas developed either in Visio's XML drawing format or directly in a canonical ER-XML format developed by us (an XML Schema specification that allows a standard representation of an ER schema). This can be extended in the future to allow additional XML representations for ER or UML (since the core constructs and their purpose are similar). All options allow users to specify entity classes (including strong and weak classes), relationships (interaction, inclusion, etc.), and constraints.

We briefly describe the trigger generation strategy employed. Depending on the nature of the constraint summarized in Table 12, the firing event is set to INSERT, DELETE or UPDATE. Since the integrity constraints must be checked prior to any possible violations, we specify it be checked BEFORE the database event takes place. A minimum cardinality specification of > 0, implies the need for a trigger fired on deletions, while a specified maximum cardinality (other than simply "many") requires checking counts before tuple insertion. We assume identifier values are unchangeable. If the database allows updates to primary key values, then update event triggers must be used for both of the previous cases as well. Triggers checked on update are also needed when a predicate is specified for the constraint. Our current system assumes that all constraint vi-

```
CREATE OR REPLACE TRIGGER ⟨Trigger Name⟩
    BEFORE ⟨Firing Event as described in Table 12⟩
    ON ⟨Affected Relation⟩
    DECLARE
       ⟨Variables including Cardinality Limits⟩
    BEGIN
                ...
       ⟨Perform count based on SQL clause as described in section 4⟩
       ...
       IF ⟨Cardinality Limits are Violated⟩ THEN
          ⟨Perform Database Actions to Manage Violation⟩
       END IF;
       ...
    END;
/
```

Figure 9: Generalized Trigger Template (Oracle)

olations must be prevented (hence an application error is raised to block the database operation). However, the user may choose to specify different actions that could be taken, including simply generating warnings, logging the violations, and so on (discussed in depth in active database literature (Paton and Díaz 1999). We leave the handling of violationaction refinements for future work.

Figure 9 shows a generalized trigger template. While the automated generation may seem straightforward, we encounter locking granularity issues for multi-user environments. Typically, a transaction that only inserts a new tuple would not obtain an exclusive table-level lock (since it is overly restrictive). This can cause constraint violations. Using our earlier example of, "any supplier and project combination can appear in the SubmitBids relationship between [h .. k] times", let's assume we were trying to enforce the upper bound of k where currently there were k-1 entries for an $\langle s, j\rangle$ pair. Suppose two transactions $t1$ and $t2$ both attempt to insert a new instance of $\langle s, j\rangle$ with some $p1$, $p2$, where $p1 \neq p2$, and the associated triples were new to the relation (i.e., both ⟨s, j, p1 ⟩ and $\langle s, j, p2\rangle \notin$ SubmitBids). Given that each would not see the uncommitted inserts of the other transaction, they would both count k-1 entries, and permit the insertion to proceed (since the insertion likely would not violate any other database constraints), resulting in k+1 instances of the $\langle s, p\rangle$ pair (a violation). As can be seen, this requires the use of an exclusive lock on affected table which leads to inefficiencies.

For special classes of constraints, notably those applied at the set level, another approach is to maintain a constraint metadata table (CMT). The CMT would contain the type, cardinality limits, and current counts for the constraint in question. Instead of performing the count on the data table (e.g., SubmitBids) each time an insert took place, the trigger could look up the CMT (the tuple corresponding to the constraint in question would be locked) and suitably adjust the attribute for the current count. The problem with applying this to instance-level constraints (that are not restricted to a small set of entities), is lack of scalability. There could be a large number of suppliers (for example) and we would not wish to store, one entry in the CMT per supplier. Pre-sorting or an index variant (with a supplier-count pair) could make this

option more efficient at an instance level, but we feel that an effective and generalizable solution would require native DBMS support for cardinality constraint enforcement. Further, while some would argue for application level support instead, we feel instead that it would be more efficient to perform the operations at the database level (rather than transmit the data to the application to do the check). In the case of the two transactions *t1* and *t2* described previously, native DBMS support could permit better scheduling of potentially conflicting transactions and re-use of knowledge from recently performed counts.

6 Conclusions and Contributions

An information system may need to manage a large number of rules from governmental, industry and organizational requirements. Rules assert business structure, can influence organizational behavior (Hay et al. 2000) and describe a state of affairs that the business wants to exist (Morgan 2002). Understanding rules, including cardinality, and modeling them in the conceptual schema in a visible manner is crucial, because if they are missed at this stage, they may never be enforced (or be applied inconsistently) when the database is implemented (Shao and Pound 1999). In addition to the value of establishing completeness of a rule framework from a research perspective, we also see utility from the perspective of model-driven architecture (MDA) (OMG 2001). The connection between conceptual modeling and MDA is important in information systems development (Kop et al. 2007), and approaches like MDA benefit from standardized constraint representation (J. B. Warmer and A. G. Kleppe 2003). Without classification, there are limitations to enabling a standardized syntax to represent constraint types and the subsequent mapping into code. Establishing completeness likewise benefits CASE tools incorporating such frameworks.

We have discussed an approach to establishing completeness of cardinality constraint frameworks. In doing so, we added a constraint category (set-level), orthogonal to existing types of cardinality rules not previously seen in literature. Modeled constraints may be represented in the conceptual schema by the analyst using a variety of syntactical approaches for ER or UML. Subsequently, a well-defined mapping can be used to come up with the corresponding implementation code.

The basic approach to supporting constraint implementation is by translating the specific constraint logic into SQL and embedding that within triggers or procedures. Given an ER schema and the associated constraints, the relevant count of data values can be checked against the constraint limits. We described a prototype currently under development to automatically translate modeled constraints from the conceptual design level into database triggers. Such checks can be implemented using Oracle's PL/SQL or SQL Server's Transact-SQL. Our efforts also lead us to recommend better DBMS support for cardinality to improve the efficiency of the constraint checking triggers.

We feel our approach shows the value of concise semantic-based rule annotation schemes.

Establishing completeness supports constraint incorporation into CASE tools for productivity gains during the development lifecycle. The platform independent conceptual taxonomies (which can be represented using a variety of syntaxes including XML or the annotation scheme shown in Appendix B) support distributed development while fitting well into the service-oriented computing paradigm. Although we used the ER model in describing constraints in this paper, the extension to UML and other modeling languages is straightforward. We feel future research directions could test whether similar approaches to completeness can be extended to other kinds of database rules, including temporal and spatial integrity constraints.

References

Atzeni P., Chen P. P. (1981) Completeness of query languages for the entity-relationship model.

In: Proceedings of the Second International Conference on the Entity-Relationship Approach to Information Modeling and Analysis. North-Holland Publishing Co., pp. 109–122

Bajaj A., Ram S. (2002) SEAM: A State Activity Entity Model for a Well Developed Workflow Development Methodology. In: 14 (2), pp. 415–431

Campbell D. M., Embley D. W., Czejdo B. D. (1985) A relationally complete query language for an entity-relationship model. In: Proceedings of the Fourth International Conference on Entity-Relationship Approach. IEEE Computer Society, pp. 90–97

Chen P. P.-S. (1976) The entity-relationship model—toward a unified view of data. In: ACM Transactions on Database Systems (TODS) 1(1), pp. 9–36

Codd E. F. (1972) Relational completeness of data base sublanguages. IBM Corporation

Currim F. (2004) Spatio-Temporal Set-Based Constraints In Conceptual Modeling: A Theoretical Framework and Evaluation. PhD thesis, University of Arizona, Tucson, AZ

Currim F., Neidig N., Kampoowale A., Mhatre G. (2010) The CARD System. In: Proceedings of the Twenty-ninth International Conference on Entity-Relationship Approach. Springer, pp. 433–437

Currim F., Ram S. (2012) Modeling spatial and temporal set-based constraints during conceptual database design. In: Information Systems Research 23 (1), pp. 109–128

Date C. J. (2000a) An introduction to database systems. In: Boston: Pearson/Addison Wesley 27(983), p. 22

Date C. J. (2000b) What not how: the business rules approach to application development. Addison-Wesley Professional

Ferg S. (1991) Cardinality Constraints in Entity-Relationship Modeling.. In: ER, pp. 1–30

Halpin T., Morgan T. (2010) Information modeling and relational databases. Morgan Kaufmann

Hay D., Healy K. A., Hall J., Bachman C., Breal J., Funk J., Healy J., McBride D., McKee R., Moriarty T. (2000) Defining business rules-what are they really. In: Final Report

ISO (1987) Information processing systems–concepts and terminology for the conceptual schema and the information base

Kop C., Mayr H. C. (1998) Conceptual predesign bridging the gap between requirements and conceptual design. In: Proceedings of the 3rd International Conference on Requirements Engineering (ICRE’98). IEEE, pp. 90–98

Kop C., Mayr H. C., Yevdoshenko N. (2007) Requirements Modeling and MDA–Proposal for a Combined Approach. In: Advances in Information Systems Development, pp. 191–201

Lenzerini M., Santucci G. (1983) Cardinality Constraints in the Entity-Relationship Model. In: ER, pp. 529–549

Liddle S. W., Embley D. W., Woodfield S. N. (1993) Cardinality constraints in semantic data models. In: Data & Knowledge Engineering 11(3), pp. 235–270

Mayr H. C., Kop C. (2002) A User Centered Approach to Requirements Modeling. In: Proceedings of the Modellierung 2002. Lecture Notes in Informatics P-12 (LNI), GI-Edition, pp. 75–86

McAllister A. (1998) Complete rules for n-ary relationship cardinality constraints. In: Data & Knowledge Engineering 27(3), pp. 255–288

Morgan T. (2002) Business rules and information systems: aligning IT with business goals. Addison-Wesley Professional

Navathe S., Batini C., Ceri S. (1992) Conceptual Database Design–an Entity-Relationship Approach. In: Redwood City: Benjamin Cummings

OCL (2014) Object Constraint Language (OCL), Version 2.4

OMG (2001) Model Driven Architecture

OMG (2015) Unified Modeling Language (UML), version 2.5

OMG (2017) Semantics of Business Vocabulary and Business Rules, version 1.4

Paton N. W., Díaz O. (1999) Active database systems. In: ACM Computing Surveys (CSUR) 31(1), pp. 63–103

Ram S., Khatri V. (2005) A comprehensive framework for modeling set-based business rules during conceptual database design. In: Information Systems 30(2), pp. 89–118

Ramesh V., Ram S. (1997) Integrity constraint integration in heterogeneous databases: An enhanced methodology for schema integration. In: Information Systems 22(8), pp. 423–446

Ross R. (2005) Business Rule Concepts: Getting to the Point of Knowledge. Business Rule Solutions Inc.

Shao J., Pound C. (1999) Extracting business rules from information systems. In: BT Technology Journal 17(4), pp. 179–186

Simsion G. (2001) Data Modeling Essentials: Analysis, Design, and Innovation Scottsdale. In: AR, Coriolis

Storey V. C., Yang H.-L., Goldstein R. C. (1996) Semantic integrity constraints in knowledge-based database design systems. In: Data & Knowledge Engineering 20(1), pp. 1–37

Sweeney L. (2002) k-anonymity: A model for protecting privacy. In: International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(05), pp. 557–570

Thalheim B. (1992) Fundamentals of cardinality constraints. In: International Conference on Conceptual Modeling. Springer, pp. 7–23

Thalheim B. (1996) An overview on semantical constraints for database models. In: Proceedings of the 6th International Conference Intellectual Systems and Computer Science

Umanath N. S., Scamell R. W. (2007) Data Modeling and Database Design. Thomson Course Technology

Vöhringer J., Mayr H. C. (2006) Integration of schemas on the pre-design level using the KCPM-approach. In: Advances in Information Systems Development. Springer, pp. 623–634

Von Halle B. (2001) Business rules applied: building better systems using the business rules approach. Wiley Publishing

Wand Y., Storey V. C., Weber R. (1999) An ontological analysis of the relationship construct in conceptual modeling. In: ACM Transactions on Database Systems (TODS) 24(4), pp. 494–528

Wand Y., Weber R. (2002) Research commentary: information systems and conceptual modeling—a research agenda. In: Information systems research 13(4), pp. 363–376

Warmer J. B., Kleppe A. G. (2003) The object constraint language: getting your models ready for MDA. Addison-Wesley Professional

Warmer J., Kleppe A. (1999) The Object Constraint Language. Addison–Wesely

Weber R. (1996) Are attributes entities? A study of database designers' memory structures. In: Information Systems Research 7(2), pp. 137–162

Vol. 13, No. 2 (February 2018). DOI:10.18417/emisa.si.hcm.23

Appendix A : ER Model Syntax

Symbol	Construct	Description
	Entity Class	A set of entities for which common properties (attributes) are to be modeled
	Regular Attribute	Properties shared by all members of an entity class.
	Multi-valued Attribute	A single entity may have more than one value for such attributes
	Identifying Attribute	An attribute that distinguishes one member entity from another
	Partial Identifier	An attribute in a weak entity class used in conjunction with identifiers from strong entity classes for distinguishing member entities
	Weak Entity Class	An entity class dependent on another (strong) entity class for its existence and part or all of its identifying attribute
	Interaction Relationship	Association between members of one or more entity classes
(from weak entity class)	Identifying Relationship	The arrow head denotes that a weak entity class depends on the relationship to determine its identifying classes
superclass D superclass O	Inclusion Relationship D : disjoint subclasses O : overlapping subclasses	Relationship defining a generalization / specialization relationship between members of entity classes

Appendix B : Annotation Syntax in Backus-Naur Form

⟨Construct Cardinality⟩	::=	⟨Class Constraint⟩ | ⟨Attribute / Domain Constraint⟩ | ⟨Interaction Relationship Constraint⟩
⟨Class Constraint⟩	::=	CARD-C (⟨Entity Class⟩)
⟨Attribute / Domain Constraint⟩	::=	CARD-A (⟨Entity Class⟩, Attribute) | CARD-D (⟨Entity Class⟩, Attribute)
⟨Attributes⟩	::=	Attribute, ⟨Attributes⟩ | Attribute
⟨Interaction Relationship Constraint⟩	::=	CARD-R-PT (⟨RE⟩) | CARD-R-PT-SET (⟨RE⟩) | CARD-R-PJ (⟨RE⟩) | CARD-R-CO (⟨RE⟩, ⟨Entity Classes⟩) | CARD-R-CO-SET (⟨RE⟩, ⟨Entity Classes⟩)
⟨RE⟩	::=	Relationship, ⟨Entity Classes⟩
⟨Entity Classes⟩	::=	⟨Entity Classes⟩, ⟨Entity Class⟩ | ⟨Entity Class⟩
⟨Entity Class⟩	::=	Entity Class | Role | Role:⟨Class⟩
⟨Class⟩	::=	⟨Entity Class⟩ | Interaction Relationship Class
⟨Card⟩	::=	⟨min⟩:⟨max⟩ | ⟨min⟩. . . ⟨max⟩ | set-builder definition
⟨min⟩	::=	Natural
⟨max⟩	::=	Natural | M