Universidad Politécnica de Madrid
Facultad de Informática

Free University of Bozen-Bolzano
Faculty of Computer Science

Extracting and Materializing the Conceptual Schema from a Relational Database

by

Lina Lubyte

A Thesis submitted for the degree of
European Masters in Computational Logic

September 2006

This work was carried out under the supervision of

Prof. Enrico Franconi

Prof. Asunción Gómez Pérez

Abstract

A number of important database problems have been shown to have improved solutions by using a conceptual model (an ontology) to provide *precise semantics* for a database schema [5]. At its best, such semantics of the data is captured by some kind of *semantic mapping* between the database schema and the conceptual model. In this thesis, we study the problem of extracting and materializing the conceptual schema of a relational database designed according to Entity-Relationship techniques.

To uncover the connections between the schema and a formal conceptual model, in the extraction process we rely on standard database design principles from ER diagrams. Indeed, at the first step of our work, we revisit these principles thus obtaining ER-to-Relational methodology that defines correct relational representations from ER model to relational model by fully exploiting the *constraints* on ER objects in order to capture the information, including primary and foreign keys, uniqueness constraints, null values, domain constraints, etc.

At the second step, which is the goal of our work, we devise the process for extracting ER model constructs together with a set of views, under the assumption that the input database was designed with the proposed ER-to-Relational methodology. The semantic mapping between the database schema and the conceptual model is thus expressed, where every view, defined over the actual data, corresponds to a unique ER construct. We theretofore derive classification schemes for relations necessary for the extraction process. In addition, we present algorithms for deciding on constraints for the ER constructs. Furthermore, we clearly identify cases in which human input is required.

To demonstrate the cause of using the conceptual schema as a global view over the data sources, we present a query answering system that provides an expressive DL based language, *DLR-Lite*, to formalize the obtained conceptual schema and allows for answering queries over relational data sources that are described by means of conceptual schema. We verify the results with a case study.

Acknowledgements

I would like to express my gratitude to a number of people without whose advise and encouragement this thesis would not have been possible.

First, I would like to thank my supervisors, Prof. Enrico Franconi and Prof. Asunción Gómez Pérez for their understanding and valuable suggestions during my research.

I wish to thank again Enrico Franconi for guiding through the European Masters Program at FUB and special thank to Federica Cumer for explaining and helping with bureaucratic rules of the University. I would also like to thank Susana Muñoz Hernández for always helping and answering questions concerning life in UPM.

I want to thank all my friends in Bolzano, with which I shared these years, in particular, Juozas, Linas, Rita, Gytis, Barbara and Martin, for the great time I had with you throughout my Masters Program at Free University of Bolzano. Thank you for trips to mountains, late tea drinking evenings and home atmosphere. I will always remember those days as among the most beautiful of my life! Additionally, a great thank to the European Masters company in Madrid - Zivile, Stefano, Luciana, Arsen, Atif, Miao and Wiwi - for traditional food parties, support and friendship.

My deepest gratitude goes to my family, my father and mother, Steponas and Silvestra, and my younger sister and brother, Lijana and Modestas, for always being with me and understanding me for not having spent so much time with them.

Last, but not least, I want to thank my beloved Daniel for being near me in this last year of work and study, for his encouragement and given happiness. Without his support and patience this work would have never come into existence. Thanks also to his family for their love and great help with italian.

Contents

List of Algorithms

List of Figures

Chapter 1

Introduction

The idea of using ontologies as a conceptual view over data repositories is becoming more and more popular. For example, in Enterprise Application Integration [27], Data integration [30], and Semantic Web [26] (for a survey see [41]), the intensional level of the application domain can be profitably represented by an ontology, so that clients can rely on a shared conceptualization when accessing the services provided by the system.

The purpose of an ontology in computer science is to *formally and unambiguously* describe the relevant notions of a domain. This formalization of the terminology then constitutes the basis for a shared and generally accepted understanding of this domain. In particular, ontologies can be used to support human understanding and communication by making the notions used in such communication precise [29]. Additionally, ontologies facilitate content-based access, communication, and integration across different information systems because they assign a precise *meaning* to the data stored in information systems.

Figure 1.1 depicts the role of an ontology (the blue top layer) with respect to the information structures (the middle layer) and to data (the bottom layer, modeled itself by the information structures). In this sense, an ontology is a formal conceptualization of the information world, specifying a set of *constraints* - which declare what should necessarily hold for any data organized according to that information structure.

1.1 Motivation

In our analysis, we focus on relational databases, as kind of information structures, and the notion of ontologies as means to organize *information*. In this setting the ontology constitutes a conceptual layer which represents the conceptualization of the domain of interest, whereas the relational database constitutes a data layer containing data related to the domain of interest.

1.1.1 Ontologies from Database Schema

As ontologies provide a conceptual view of the application domain, there is a recent trend in enterprise integration systems to use ontologies for providing a global, unifying

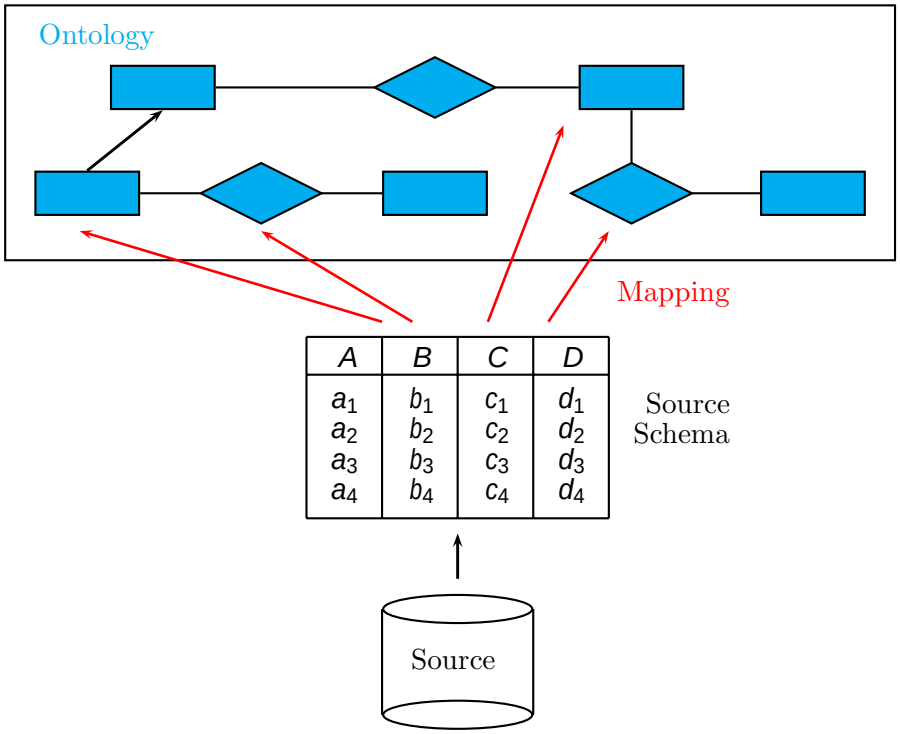


Figure 1.1: The role of an ontology

view on the data. If such an ontology exists, databases can be designed to conform with it and the ontology can be employed for navigational (and reasoning) purposes when accessing the data.

Suppose the goal is to design an ontology that is to be used as a unifying view in enterprise integration. Typically, there will be a number of database systems that are used in the enterprise to be modeled. Since databases play a very central role in the information infrastructure of enterprises, the structure of the database contains a lot of information about the enterprise. This information is explicit in the conceptual schema. Therefore, it is a natural approach to start the ontology design process with extracting information from the conceptual database schemas, i.e., translating such schemas into an initial ontology that can then be corrected, reconciled, and completed manually. The main reasons why this approach is advantageous include the following:

- Databases play a very prominent role in information system management in most well-founded enterprises. They contain considerable information about domains of interest. For example, an online book store company must have comprehensive information about books, book readers, and possibly also the

statistic data about book purchase transactions. Database systems have been designed, used, maintained, and redesigned, over time, until they perfectly reflect the real needs. Therefore, it is often desirable to make use of this existing information when an ontology about the same domain is being designed. An obvious advantage is that the ontology designer need not author every concept description from scratch, and that he obtains, with minimum effort, a decent ontology that captures the knowledge about the domain.

- Once a core ontology is obtained from a database schema, the ontology developer could do one of the following to enrich the core ontology. The first scenario would be that the ontology developer extends the core ontology using a more powerful ontology language to cover all the semantic subtleties and to have a larger vocabulary coverage. In the second scenario, the ontology developer could integrate the generated core ontology with a large, autonomously produced ontology. For instance, the core ontology about an online book store could suitably be integrated with a generic ontology about books and another about online stores. In both situations, query answering to the database can be made more powerful when realized as ontology-based information access using the extended ontology. In these contexts, query answering can take advantage of a more powerful conceptual view over data repositories and efficient query handling of relational database management system (RDBMS).

Thus, the ontology designer wants to create an ontology about an enterprise domain that is already (partly) described in the form of a database schema. Instead of constructing the ontology from scratch, he wants automatic support for converting the database schema into an initial ontology.

To automatically generate an initial ontology from existing relational database, a reasoner takes a database schema as input and generates an ontology as output. To formalize the conceptual schema of a database, we use the formalism based on Description Logics (DLs) [6]. Specifically, we have extended the DL *DL-Lite* [9] with the ability to support n-ary relationships, obtaining the DL *DLR-Lite*. Such a formalism is expressive enough to capture basic Entity-relationship or UML class diagrams, while allowing query answering that fully takes into account the constraints in the conceptual schema and is still tractable (i.e., polynomial) in the size of the data [28].

The reasoner should extract as much information from the database as possible. However, the information contained in the database itself is usually *incomplete*, due to the poor database modeling methodologies or mistakes made by database designers. For this reason, we revisit the methodology of translating the conceptual schemas to relational schemas, by fully exploiting the conceptual schema (ontology) knowledge in order to capture the information, including primary and foreign keys, uniqueness constraints, null values, domain constraints, etc. In addition, the careful study of relating the conceptual model with the relational model underlie the heuristics used in ontology creation process. Namely, we use ideas of standard relational schema design from ER diagrams in order to craft heuristics that systematically uncover the connections between the constructs of relational schemas and those of ontologies.

1.1.2 Information Access

An important goal in many applications is to access information with the mediation of an ontology, which models information domain constraints at the conceptual level. Once the ontology is extracted from the database schema, the problem of accessing information via ontologies becomes very relevant to convey semantic information during the query formulation. The ontology defines a vocabulary which is richer than the logical schema of the underlying data, thus the user would prefer to query the database using the rich vocabulary of the ontology. Moreover, this allows users (or applications) to formulate queries also in the case of schema mismatch, since the information system is accessed using the common and agreed-upon ontology vocabulary.

Another functional aspect of the query support task is so called *intentional navigation* [21]. Intensional navigation can help a less skilled user during the initial step of query formulation, thus overcoming problems related with the lack of schema comprehension and so enabling her/him to easily formulate meaningful queries. Queries can be specified through an iterative refinement process supported by the ontology through intensional navigation. The user may specify her/his request using generic terms, refine some terms of the query or introduce new terms, and iterate the process. Moreover, users may explore and discover general information about the domain without querying the database, giving instead an explicit meaning to a query and to its subparts through classification. Summing up, the query is expressed using the ontology vocabulary and it is evaluated by means of reformulation or direct evaluation over the underlying database.

Technology

The scenario described above is based on a unified view of data, called *mediated or global ontology*, and a module, called *mediator*, that collects and combines data extracted from the sources, according to the structure of the mediated schema. The crucial aspect in the design and the realization of mediators is the specification of the relationship between the sources and the mediated schema (i.e., mapping between the conceptual and the data layer). Two basic approaches have been proposed in the literature, called global-as-view (or simply GAV) and local-as-view (or simply LAV) [30]. In the GAV approach, a view over the sources is associated to each element of the global schema, describing how to populate such an element using the data at the sources. In this thesis we adopt GAV approach, thus, the vocabulary over the data layer can be seen as a set of (materialized) views over the vocabulary of the ontology. However, in this case we have to solve the problem of view-based query processing. The problem requires to answer a query posed to a database - the one defined by the ontology - only on the basis of the information in a set of (materialized) views, which are again queries over the same database. In the process, the information contained in the ontology should be of course taken into account.

1.2 Related Work

1.2.1 Data Integration

Notice that the setting described above on information access can be immediately extended to consider several independent DBMSs constituting the data layer. In this sense, such a scenario generalizes to the presence of ontologies typical data integration scenarios studied in the database literature [30, 32].

A data integration system provides a uniform query interface to a multitude of autonomous, distributed, and heterogeneous data sources, which may reside within a local database or on distributed databases. The idea is to free the user from having to deal with several sources to have a complete answer to his queries. Queries are formulated using a uniform query language against a uniform *virtual view* of the data (mediator), instead of querying different data schemas. The querying system translates the query against the view into queries against the sources, retrieves results on data sources, and finally represents them to the user [30].

1.2.2 Answering Queries Using Views

An important connection, as already noted before, is the research on answering queries using views (see [25] for a survey). View-based query processing is the problem of computing the answer to a query posed on a set of views [40, 1]. There are two basic approaches to view-based query processing. The first is the *query rewriting* approach, which is based on the idea of first reformulating the query in terms of the views and then evaluating the rewriting over the view extensions. The other, *query answering* approach, takes a more direct route, trying to compute the so-called *certain tuples*, i.e., the tuples satisfying the query in all databases consistent with the views, on the basis of the view definitions and the view extensions.

The mappings between the ontology and the database in our work are defined as a set of view definitions, where each view corresponds to the element in an ontology. What we are seeking at the end is answers to queries phrased in terms of the ontology predicates, but rephrased in terms of relational tables, where the data instances reside - which is exactly the problem of query answering by rewriting using views.

1.2.3 Reverse Engineering of Relational Databases

Other potentially relevant work includes *database reverse engineering*, which is the process of two distinct steps: i) eliciting the data semantics from the system, and ii) expressing the extracted semantics with a high level data model [24]. Many approaches to database reverse engineering overlook the first step by assuming that some of the data semantics is known (the survey of database design transformations using the ER model is presented in [18]). Sophisticated algorithms and approaches to this have appeared in the literature over the years (e.g., [33, 12, 3]).

In our work, in the view construction and the ontology extraction process, we use the idea based on primary key and foreign key correlations, which, in a similar principle, is used in [33] and with a more broad view in [1].

1.3 Contributions of the Thesis

Without going into further details at this point, we summarize the contributions of this work:

- We revise the mapping methodology of ER model to relational model in order to represent ER objects in a database without loss of information. We provide a complete methodology for ER-to-Relational mapping, thus obtaining the data sources that are complete w.r.t. the ER model describing them.
- Assuming to have a database designed with developed ER-to-Relational methodology (i.e., with complete data sources), we define the process of extracting from it a set of views together with ER components, in such a way that every view corresponds to a unique ER component. For this, we devise the classification on relations based on correlations between keys and foreign keys. Additionally, we propose algorithms for deciding on cardinality and participation constraints for ER components.
- Having the setting described above, we present a mechanism for answering queries over relational data sources (i.e., views) that are described by means of an ER schema. We provide an expressive schema language which covers features of ER diagrams, as well as ontology languages like Description Logics. We present the query rewriting process, which is independent of the actual data in the information sources, since it uses the information from the conceptual schema only. Finally, we display all the process with a detailed case study.

1.4 Outline of the Thesis

The rest of the thesis is structured as follows. In chapter 2 we present technical background and notation, surveying the literature about Entity-relationship model and basic concepts of the relational model. In Chapter 3 we develop ER-to-Relational methodology, that fully exploits the ER model knowledge in order to capture as much information as possible, and present the results of relation between conceptual and relational constructs. In chapter 4 we first devise a classification on relations that is based on the appearance of primary and foreign keys in a relation. Further we present the process of defining views and identifying corresponding ER components, as well as present the algorithms deciding on constraints for these components. Chapter 5 presents a mechanism for answering queries over relational data sources using their conceptual schema. First, the framework of the system is provided. Then we describe the queries over this system and display the query rewriting process. A detailed case study finalizes the chapter. Finally, we conclude in Chapter 6.

Chapter 2

Formal Preliminaries

In this chapter we define the concepts used throughout this thesis. We first provide all the major concepts of the semantic data model being used for conceptual domain modeling. In databases, the best known such model is Entity-Relationship (ER) data model. Thus the ER data model will be described in considerable detail. Secondly we define a brief formal description of the relational model with integrity constraints. Then we illustrate the classes of queries that we deal with in the following and that are of interest in the context of this thesis. Finally, we give some formal definitions on views in relational databases. For further background we refer to [2].

2.1 The Entity-Relationship Model

Entity-Relationship (ER) is the most widespread semantic data model, and it has become a standard, extensively used in the design phase of commercial applications. The ER model was introduced in [11], with minor variants and extensions proposed over the years (e.g., [36, 39, 7]). The overriding emphasis in ER modeling is on simplicity and readability. The goal of conceptual schema design, where the ER approach is most useful, is to capture real-world data requirements in a simple and meaningful way that is understandable by both the database designer and the end user.

In the following subsection we present the basic elements and notions of ER model. More details can be found in [7, 37].

2.1.1 Basic Constructs

The basic elements of the ER model are entities, relationships and attributes. An *entity set* (or simply *entity*) denotes a set of objects, called its *instances*, that have common properties. Elementary properties are modeled through *attributes*, whose values belong to one of several predefined domains, such as `Integer`, `String` or `Boolean`. Properties that are due to relations to other entities are modeled through the participation of the entity in relationships. A *relationship set* (or simply *relationship*) denotes a set of tuples (also called its instances), each of which represents an association among a different combination of instances of the entities that participate in the

relationship. Since each entity can participate in a relationship more than once (e.g., a company may be the recipient or sender in a "supply" relationship), the notion of *ER-role* is introduced to represent such a participation, and to which a distinguishing identifier within the relationship is assigned. The *arity* of a relationship is the number of its ER-roles. We assume that, for each relationship of arity n, the identifiers $1, \ldots, n$ are assigned to the roles of the relationship. Note that there can be a relationship connecting an entity to itself with different ER-roles. Such relationships are called *recursive* relationships.

An entity B is said to be a specialization/IS-A of another entity A, if all the instances of B are also instances of A. This induces an inheritance of the attributes of an entity to its sub-entities, and of the roles of a relationship to its sub-roles. The ER schema produced as a result of ER modeling is usually represented in a graphical notation, which is particularly useful for an easy visualization of the data dependencies. In the commonly accepted notation, entities are represented as boxes, whereas relationships are represented as diamonds. An attribute is shown as a circle attached to the entity for which it is defined. ER-roles are graphically depicted by connecting the relationship to the participating entities, and (possibly) labeling the edges with the corresponding role identifier. Regarding the graphical representation of IS-A relationship, there are many alternative notations. In this document, for simplicity, the subclasses are attached by lines to a diamond that represents the IS-A relationship, which is connected to the superclass.

Cardinality constraints can be attached to the ER-role in order to restrict the number of times each instance of an entity is allowed to participate via that ER-role in instances of the relationship. Such constraints can be used to specify both existence dependencies and functionality restrictions [15]. They are often used only in a restricted form, where the minimum cardinality is either 0 or 1 and the maximum cardinality is either 1 or n (sometimes denoted as ∞). Cardinality constraints in the form considered here follow this restriction and have been subsequently studied in [23, 31, 19, 42, 10].

The constraints to sub-entities in IS-A relationship are applicable only if specialization consists of at least two sub-entities. We consider two constraints: *disjointness* and *covering*. The former specifies that the sub-entities of the specialization must be disjoint. This means that an entity can be a member of at most one of the specialization entities in IS-A relationship. If sub-entities are not constrained to be disjoint, their entities may *overlap*. That is, the same entity may be a member of more than one sub-entity of the specialization. The latter, covering constraint specifies that every entity in the superclass must be a member of at least one subclass in IS-A relationship. Graphically, we display these constraints by placing the word "disjoint" or "covering" close to the IS-A diamond.

2.2 Relational Concepts

2.2.1 Background

In the following definition, uppercase letters (which may be subscripted) from the end of the alphabet, such as X, Y, Z, will be used to denote sets of attributes, while those from the beginning of the alphabet, such as A, B, C, will be used to denote single attributes. The cardinality of a set X is denoted by $|X|$.

Definition 2.2.1 (Basic relational database concepts)**.** A *relation schema* R, denoted by $R(X)$, is the name of the relation R and a finite set of distinct attributes $X = \{A_1, \ldots, A_m\}$. Each attribute is the name of a role played by some domain D in the relation schema $R(X)$. Thus, the *domain* of an attribute A_i, $(1 \leq i \leq m)$, denoted by $dom(A_i)$, is the set of all possible values of A_i. Two attributes are said to be *compatible*, if they are associated with the same domain. A relation schema is used to describe a relation.

A *relation* r of the relation schema $R(X)$ is a finite set of tuples over R. Each *tuple* is an ordered list of n values $t = \langle v_1, \ldots, v_n \rangle$, where each value v_i, $1 \leq i \leq n$, is an element of $dom(A_i)$ or is a special *null* value, representing the values of attributes that may be unknown or may not apply to a tuple. Tuples t_1 and t_2 are *equal*, if and only if they have the same attributes $A_1, \ldots, A_m$, and for all i $(1 \leq i \leq m)$, the value v_1 of A_i in t_1 is equal to the value v_2 of A_i in t_2.

We consider to have an infinite, fixed database domain Δ whose elements can be referenced by constants $c_1, \ldots, c_n$ under the *unique name assumption*, i.e., different constants denote different real-world objects. Then a *relational schema* $\mathcal{R}$ is a pair $(\mathbf{R}, \Sigma)$, where $\mathbf{R}$ is a finite set of relation schemas $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$, Σ is a set of *integrity constraints* expressed on the relations in $\mathbf{R}$, i.e., assertions of the relations in $\mathbf{R}$ that are intended to be satisfied by database instances.

A *database instance* (or simply *database*) $\mathcal{D}$ for a schema $\mathcal{R} = (\mathbf{R}, \Sigma)$ is a set of facts of the form $r(c)$, where r is a relation of arity n in $\mathbf{R}$ and c is an n-tuple of constants of Δ. We denote as $r^{\mathcal{D}}$ the set $\{c | r(c) \in \mathcal{D}\}$. A database $\mathcal{D}$ for a relational schema $\mathcal{R}$ is said to be *consistent* with $\mathcal{R}$ if it satisfies as constraints expressed on $\mathcal{R}$.

Definition 2.2.2 (Relational algebra concepts)**.** If $Y \subseteq X$ is an attribute set, and t a tuple over R, we denote by $t[Y]$ the restriction of t to Y. The *projection* of a relation r over $R(X)$ onto Y is denoted by $\pi_Y(r)$ and generates a relation associated with attribute set Y, that is equal to $\{t[Y] | t \in r\}$.

Let r_1 and r_2 be two relations over relation schemas $R_1(X_1)$ and $R_2(X_2)$, respectively. The *natural join* of r_1 and r_2 is denoted $r_1 \bowtie r_2$, and generates a relation associated with attribute set X_1X_2, that is equal to $\{t | t[X_1] \in r_1, t[X_2] \in r_2\}$.

2.2.2 Integrity Constraints

The schemas alone are not sufficient to describe a database. Databases contain knowledge as well as data. The database schema - how the data is organized - is knowledge, which yields constraints on the form the data must take. The relationships between data that must hold, such as functional and inclusion dependencies, are knowledge

as well [22]. It is beneficial for a database to store explicitly its knowledge, in addition to its data. Most of the database's knowledge is captured and stored via *integrity constraints*, statements about what are the legal states and transitions of the database.

The most important of these constraints are functional and inclusion dependencies, defined in the following subsection.

Functional and Inclusion Dependencies

Definition 2.2.3 (Functional dependency)**.** A *functional dependency* (FD) over a relation schema $R(X)$ is a statement of the form $R : Y \rightarrow Z$, where $Y, Z \subseteq X$ are sets of attributes. An FD $R : Y \rightarrow Z$ is satisfied in a relation r over $R(X)$ if whenever two tuples have equal Y-values, they also have equal Z-values. Formally, a functional dependency $R : Y \rightarrow Z$ is *satisfied* (or holds) in a relation r over $R(X)$, denoted by $r \models R : Y \rightarrow Z$, if $\forall t_1, t_2 \in r, t_1[Y] = t_2[Y] \Rightarrow t_1[Z] = t_2[Z]$.

A FD $R : Y \rightarrow Z$ is *minimal* in r, if Z is not functionally dependent on any proper subset of Y, i.e., if $W \rightarrow Z$ does not hold in r for any $W \subset Z$.

A FD $R : Y \rightarrow Z$ is *trivial* in r, if Z is a subset of Y, i.e., $Y \supseteq Z$; otherwise, it is *nontrivial*.

An FD $R : Y \rightarrow Z$ is *transitive* in r, if there exists a set of attributes W, such that $Y \rightarrow W$ and $W \rightarrow Z$ hold, and W is neither the candidate key, nor a subset of any key of $R(X)$.

Definition 2.2.4 (Inclusion dependency)**.** An *inclusion dependency* (IND) over the set of relation schemas $\mathbf{R}$ is a statement of the form $R_1[Y] \subseteq R_2[Z]$, where $R_1(X_1)$, $R_2(X_2) \in \mathbf{R}$ and Y, Z are sequences of attributes such that $Y \subseteq X_1$, $Z \subseteq X_2$, and $|Y| = |Z|$.

Let $\mathcal{D}$ be a database for $\mathcal{R}$, where $r_1, r_2 \in \mathcal{D}$ are relations over relation schemas $R_1(X_1), R_2(X_2) \in \mathbf{R}$. An inclusion dependency $R_1[Y] \subseteq R_2[Z]$ is *satisfied* (or holds) in a database $\mathcal{D}$ for $\mathcal{R}$, denoted by $\mathcal{D} \models R_1[Y] \subseteq R_2[Z]$, if $\forall t_1 \in r_1, \exists t_2 \in r_2$ such that $t_1[Y] = t_2[Z]$. (Equivalently, $\mathcal{D} \models R_1[Y] \subseteq R_2[Z]$ whenever $\pi_Y(r_1) \subseteq \pi_Z(r_2)$).

It is worth to note that Y and Z are *sequences* of attributes: the ordering of the attributes within the sets Y and Z matters. Indeed, the two following INDs are not equivalent: $\mathsf{Manager}[\mathsf{FirstName}, \mathsf{LastName}] \subseteq \mathsf{Employee}[\mathsf{FirstName}, \mathsf{LastName}]$ is different from $\mathsf{Manager}[\mathsf{LastName}, \mathsf{FirstName}] \subseteq \mathsf{Employee}[\mathsf{FirstName}, \mathsf{LastName}]$.

Keys and Foreign Keys

In this thesis we concentrate on keys and foreign keys, which are specializations of FDs and INDs respectively.

A relation is defined as a set of tuples. By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for all their attributes. Informally, a superkey is a set of attributes that determines all the attributes in a relation schema, and a key is a superkey whose set of attributes is minimal.

Definition 2.2.5 (Key)**.** A set of attributes $K \subseteq X$ is a *superkey* for a relation r over $R(X)$ if K functionally determines all other attributes of the relation, i.e., $R : K \rightarrow X$.

A set of attributes K is a *key* (or minimal key) if there exists no other superkey K' of r that is contained in K as a proper subset.

In general, a relation schema may have more than one key. In this case, each of the keys is called a *candidate key*, and an attribute belonging to a candidate key is called a *prime attribute*. It is common to designate one of the candidate keys as the *primary key* of the relation. This is the candidate key whose values are used to identify tuples in the relation. If the primary key consists of more than one column, it is called a *composite primary key*. The following are the characteristics of the primary key:

- *uniqueness* - the primary key values must identify the tuples in the relation uniquely;
- *minimality* - if the primary key is composite, no column can be eliminated from the primary key without loosing the uniqueness property.

Apart from uniqueness, the key integrity constraint, which must be enforced by the DBMS, must ensure that no component of the primary key is allowed to take the undefined null value.

One should note that at the schema level, a key is a constraint, but at the relation level, a key is a property derived from the relation. This corresponds to the distinction between a functional dependency (constraint) and the satisfaction of an FD in a relation (property).

Definition 2.2.6 (Foreign key)**.** Let $R_1(X_1), R_2(X_2)$ be relation schemas over $\mathbf{R}$, and let K be a key of $R_2(X_2)$. In addition, let $\mathcal{D} = \{r_1, r_2, \ldots, r_n\}$ be a database for $\mathcal{R}$.

A *foreign key* constraint is a specification of a set of attributes $Y \subseteq X_1$ of $R_1(X_1)$ and a key K of $R_2(X_2)$. The set of attributes Y is called a *foreign key* (FK) of $R_1(X_1)$ and it is said to *reference* the key K of $R_2(X_2)$. The foreign key constraint that Y references K is satisfied in $\mathcal{D}$ if the following condition holds: for all tuples $t_1 \in r_1$, there exists $t_2 \in r_2$ such that $t_1[Y] = t_2[K]$.

A foreign key is an IND whose right-hand side is a key. It is also called *referential integrity constraint*, or *key-based inclusion dependency*.

Apart from functional and inclusion dependencies, that describe the properties of keys and foreign keys, we investigate the following additional categories of integrity constraints:

- *attribute integrity constraints*[1] are used to specify additional restrictions on an attribute defined over a domain. We use an attribute IC to i) specify that a particular attribute may not take the undefined null value, by restricting it to be `NOT NULL`; ii) enforce the particular attribute to contain unique values, by restricting it to be `UNIQUE`;
- *tuple equality integrity constraints* specify the constraint between two relations, and namely, all values that appear in the foreign key of one relation must coincide

[1] Name of the constraint taken from Date's ICs classification [16].

with the values in the primary key of the referenced relation. These ICs are induced by the relationships in ER schema having the cardinality of $(1, 1)$ or $(1, n)$, as discussed in the next chapter. We formalize these ICs by equality on algebraic projections;

- *exclusion integrity constraints* specify the constraint between several relations, and, namely, all tuples of one relation must be disjoint with the tuples of all other relations participating in the constraint. This IC is induced by the disjointness of subclasses in IS-A relationship of ER schema. We formalize these ICs by forcing intersections on algebraic projections to be empty, i.e., $\pi_{Y_1}(r_1) \cap \ldots \cap \pi_{Y_m}(r_m) = \emptyset$, where Y_i is a set of attributes of a relation r_i, $1 \leq i \leq m$;

- *covering integrity constraints* specify the constraint between several relations, and, namely, the union of tuples of several relations form the tuples of some other relation. These ICs are induced by the subclasses that cover the superclass in IS-A relationship of ER schema. We formalize these ICs by the assertion of the form $\pi_{Y_1}(r_1) \cup \ldots \cup \pi_{Y_m}(r_m) = \pi_{Y_0}(r_0)$, where Y_i is a set of attributes of a relation r_i, $0 \leq i \leq m$.

2.2.3 Relational Normal Forms

The normal forms were initially proposed by E. Codd in [13], which he called first, second and third normal form. A normal form restricts the set of dependencies that are allowed to hold in a relation schema. The main purpose of the normal forms is to eliminate at least some of the redundancies and update anomalies that might otherwise arise [2]. Intuitively, schemas in normal form are "good" schemas. In this thesis we analyze the relations in third normal form (3NF).

Definition 2.2.7. Let R be a relation schema, F be the set of functional dependencies given to hold over R, Y be a subset of the attributes of R, and A be an attribute of R. Then R is in *third normal form (3NF)* if, for every functional dependency of the form $R : Y \rightarrow A$, one of the following statements is true:

- $A \in Y$, i.e., A is a trivial functional dependency, or
- Y is a superkey, or
- A is part of some key for R.

Any relation with a transitive dependency violates the third normal form.

This normal form is the goal in database design of this thesis.

2.3 Query Languages

In this section we briefly introduce conjunctive queries that are used in a framework of query answering. We also use SQL in the examples throughout this thesis, but since it is a standard, we skip its presentation here.

A *term* is either a variable or a constant. An *atom* is an expression $p(z_1, \ldots, z_n)$, where p is a predicate (relation) of arity n and $z_1, \ldots, z_n$ are terms. A *conjunctive query* q over a knowledge base $\mathcal{K}$ is an expression of the form

$$q(x) \leftarrow \exists y.conj(x, y)$$

where x are the so-called *distinguished variables*, y are existentially quantified variables called *non-distinguished variables*, and $conj(x, y)$ is a conjunction of atoms of the form $T(z_1, \ldots, z_n)$, where T is a relation over $\mathcal{R}$ with n attributes and $z_1, \ldots, z_n$ are terms. $q(x)$ is called the *head* of q and $\exists y.conj(x, y)$ the *body* of q.

The *answer* of a query $q(x) \leftarrow \exists y.conj(x, y)$ over a database $\mathcal{D}$ is the set $q^{\mathcal{D}}$ of tuples c of constants in a domain Δ such that when we substitute the variables x with the constants c, the formula $\exists y.conj(x, y)$ evaluates to true in $\mathcal{D}$.

A *union of conjunctive queries (UCQ)* is an expression

$$q(x) \leftarrow \exists y_1.conj_1(x, y_1) \vee \cdots \vee \exists y_m.conj_m(x, y_m)$$

where for each $i \in \{1, \ldots, m\}$ $conj_i(x, y_i)$ is a conjunction of atoms.

The answer of a UCQ $q(x) \leftarrow \exists y_1.conj_1(x, y_1) \vee \cdots \vee \exists y_m.conj_m(x, y_m)$ over a database $\mathcal{D}$ is the union of the answers of the conjunctive queries

$$\begin{aligned} q_1(x) &\leftarrow \exists y_1.conj_1(x, y_1) \\ &\vdots \\ q_m(x) &\leftarrow \exists y_m.conj_m(x, y_m) \end{aligned}$$

2.4 Views in Relational Databases

Originally, in database theory, a *view* is a virtual or logical table composed of the result set of a query. A query is a formula that specifies which data to extract from the data source. The set of answers of a query is defined relative to a database storing a set of facts that are *extensions* of some predicates. These predicates are called *extensional predicates*.

Definition 2.4.1 (Extension of predicates)**.** Let p be an extensional predicate. Its extension, denoted $ext(p)$, is a set of facts of the form $p(c)$ where c is a tuple of constants. For a set $\mathcal{P}$ of extensional predicates, we denote $ext(\mathcal{P})$ the union of the extensions of the predicates in $\mathcal{P}$.

An interpretation I is extended to a set of extensions by assigning an individual c^I of Δ^I to every constant appearing in the extensions. I is a model of $ext(\mathcal{P})$ if for every fact $p(c)$ of $ext(\mathcal{P})$, $c^I \in p^I$.

In the standard setting of relational databases, the extensional predicates are those which are defined in the schema. However, in the general case, the extensional predicates may be predicates of queries called *views*: queries whose answers have been precomputed (*materialized views*) or are known to be available from some data sources (*virtual views*).

Definition 2.4.2 (Views)**.** A view v is a query that has a definition $def(v)$ and possibly an extension $ext(v)$. Let $\mathcal{V}$ be a set of views. An interpretation I i a *model* of $\mathcal{V}$ iff I is a model of all the view predicates in $\mathcal{V}$.

Views are used for several reasons: to provide a shorthand capability, they allow the same data to be seen by different users in different ways at the same time, provide automatic security for hidden data [16]. However, the most important point of all is that views can provide *logical data independence*. We discuss this issue in more detail. One of the aspects to such logical data independence is *restructuring*. In our case it is necessary to restructure the database in the way that, although the overall information content remains the same, the *logical placement* of information changes - that is, the allocation of attributes to relations is altered in some way, e.g. we may create a view that is a join of two relations, etc.

Chapter 3

ER-to-Relational Methodology Revisited

Translating ER schemas into relational schemas involves representing the structural semantics of ER schemas using relational constructs and assigning names to relational attributes. A relational schema $\mathcal{R}$ is said to represent correctly an ER schema if ER objects can be represented in a database state associated with $\mathcal{R}$ without loss of information, and if a database state associated with $\mathcal{R}$ can contain only data on ER objects [34]. We define below correct relational representations for ER schemas, that preserve the correspondence between ER constructs and relational constructs.

3.1 Mapping ER Schemas into Relational Schemas

In the following subsections we define the mapping methodology of ER constructs to relational constructs. In order to guarantee the correct relational representation for an ER schema, we analyze all possible cases for transforming its conceptual model into a corresponding relational structure that preserve the Third Normal Form. While it is sufficient to represent ER constructs using relations and inclusion dependencies, we also use attribute integrity constraints, specifically, `NOT NULL` and `UNIQUE` constraints for representing restrictions on the allowed values for ER attributes. Furthermore, in some cases we constraint the occurrence of values of the foreign key in one relation w.r.t. the primary key in the referenced relation, by placing the tuple equality ICs. Specifically for IS-A relationships we constraint the tuples of relations with exclusion and covering ICs.

To generalize the notation, for an entity E_i we denote by $\mathsf{rel}(E_i)$ the relation representing E_i and for the relationship R_k, the relation $\mathsf{rel}(R_k)$ representing R_k. We denote by pk_i the primary key of a relation $\mathsf{rel}(E_i)$ (or pk_k the primary key of a relation $\mathsf{rel}(R_k)$) and by fk_{i_j} the foreign key of a relation $\mathsf{rel}(E_i)$ referencing the corresponding relation $\mathsf{rel}(E_j)$.

3.1.1 Mapping Entities

The correspondence between entities in the ER model and relations in the relational model is straightforward. Thus, each entity E_i is converted into a relation $\mathsf{rel}(E_i)(X_i)$ such that the following condition holds:

C1 X_i is in a one-to-one correspondence with E_i attributes, and the domain associated with an attribute A of X_i represents the value-set of the E_i attribute corresponding to A. The subset of X_i, pk_i, is declared as a primary key of $\mathsf{rel}(E_i)$, that corresponds to the identifier of E_i.

It is important to note that $\mathsf{rel}(E_i)$ does not have any indication of the relationships in which the entity type E_i participates.

3.1.2 General Conversion Rules for Unary, Binary and N-ary Relationships

There are three possible approaches for mapping relationships of ER schema to relational model [17]:

- the *foreign key* approach, where the relationship between entities is represented by including the primary key of one relation in another relation to act as a foreign key;
- the *merged relationship* approach, by merging the participating entities and the relationship in a single relation;
- the *relationship relation* or *cross-reference* approach, where the relationship between entities is represented by setting up a new relation for the purpose of cross-referencing the primary keys of the relations representing the participating entities.

Different sources suggest to apply these approaches for different situations, e.g. [7, 17] use a relationship relation option for one-to-one binary relationships with partial participation of participating entities, while [38] for all one-to-one and one-to-many binary and recursive relationships use the foreign key approach. Obviously, for all many-to-many relationships, the relationship relation approach must be applied. However, as was already mentioned before, in this thesis we consider all possible cases of mapping relationships of ER schema to tables in relational schema, and in particular, special attention is given to binary relationships, looking carefully at cardinalities on participating entities. In other words, we try to capture the decisions made by database designer, while logically mapping ER constructs to relational model. In this way, the view extraction process from relational tables can be done more precisely.

We present general rules for converting relationships[1] by using foreign key and relationship relation approaches, that will be exploited in the next section by examining every combination of cardinalities separately. Since we are constructing relational tables that preserve 3NF, the merged relationship approach is disposed.

[1]The conditions are applicable to binary, n-ary and recursive relationships.

Firstly, for the relationship conversion, it is assumed that the entities participating in the relationship have already been converted according to the rule identified above. Let R_k be the relationship between entities $E_1, \ldots, E_n$ and let each entity be represented by relations $\mathsf{rel}(E_1)(\mathsf{X}_1), \ldots, \mathsf{rel}(E_n)(\mathsf{X}_n)$, respectively. Then

C2 by applying the *relationship relation* approach, the relationship R_k is represented by a separate relation $\mathsf{rel}(R_k)(\mathsf{Att}_k)$, together with inclusion dependencies $\mathsf{rel}(R_k)\,[fk_{k_i}] \subseteq \mathsf{rel}(E_i)\,[pk_i]$, where Att_k is the union of two disjoint sets of attributes, X_k and FK_k, as defined below:

- X_k is in a one-to-one correspondence with the (local) attributes of R_k, where the correspondence is specified as in condition C1 above;
- FK_k is a set of foreign key attributes, such that $\mathsf{FK}_k = \bigsqcup_{i=1}^{n} fk_{k_i}$, where every fk_{k_i} is compatible with pk_i of $\mathsf{rel}(E_i)$, and so that attributes fk_{k_i} are pairwise disjoint, $1 \leq i \leq n$.

If every entity involved in R_k has cardinality *many*, FK_k forms the primary key, pk_k of $\mathsf{rel}(R_k)$. Otherwise, the subset of FK_k, satisfying the minimal key condition, is declared as a primary key, pk_k, of $\mathsf{rel}(R_k)$[2].

C3 by applying *foreign key* approach, the relationship R_k between entities E_i and E_j (not necessarily distinct)[3] is represented by posting the primary key of a relation, say $\mathsf{rel}(E_j)$, corresponding to the entity E_j with cardinality *many*, as a foreign key to the relation $\mathsf{rel}(E_i)$, corresponding to the entity E_i with cardinality *one* and an inclusion dependency $\mathsf{rel}(E_i)\ \bigsqcup_{j=1}^{m} fk_{i_j} \subseteq \mathsf{rel}(E_j)\,[pk_j]$ is assigned. Att_i of $\mathsf{rel}(E_i)(\mathsf{Att}_i)$ is the union of three disjoint sets of attributes, X_i, X_k and FK_i, as defined below:

- X_i is the set of (local) attributes of $\mathsf{rel}(E_i)$, as defined in condition C1 above;
- X_k is in one-to-one correspondence with the (local) attributes of R_k, where the correspondence is specified as in condition C1 above;
- FK_i is a set of foreign key attributes, such that $\mathsf{FK}_i = \bigsqcup_{j=1}^{m} fk_{i_j}$, where every fk_{i_j} is compatible with pk_j of $\mathsf{rel}(E_j)$, and so that fk_{i_j} are pairwise disjoint, m - number of posted primary keys of relations $\mathsf{rel}(E_j)$ to $\mathsf{rel}(E_i)$ by applying foreign key approach.

If R_k is one-to-one, the decision on choosing the relation for posting the primary key depends on participation constraints, as analyzed in the next subsections.

3.1.3 Mapping Binary Relationships

In this subsection we examine the possibilities to map binary relationships of ER schema to relational model, by considering all combinations of cardinalities on participating entities. We take into account the participation constraints on deciding which

[2]Primary key conditions are defined for every separate case in the next subsections.

[3]Foreign key approach is applied only for one-to-one and one-to-many binary and recursive relationships.

approaches can be applied for a particular case. In other words, we try to "catch" all decisions that a database designer can make, in order to preserve the minimal number of tables in relational schema, to avoid many null or redundant values, etc., and at the same time to preserve the 3NF on relations. Furthermore, in some particular cases, we restrict the values of foreign keys by placing `NOT NULL` and `UNIQUE` constraints, and bound the occurrence of their values w.r.t. the primary key in the referenced relation.

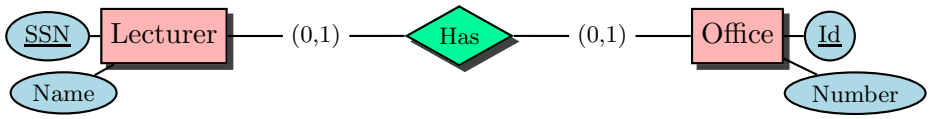
We make the following assumption on the cardinalities of participating entities: the only allowed min-max cardinalities are $(0, 1)$, $(1, 1)$, $(0, n)$, $(1, n)$. Therefore, we have 10 different cases (symmetric cases are excluded), which are all captured below. Again, let entities E_i and E_j participate in a binary relationship R_k and let E_i and E_j be represented by relations $\mathsf{rel}(E_i)(\mathsf{X}_i)$ and $\mathsf{rel}(E_j)(\mathsf{X}_j)$, respectively, that will sometimes be referred to tables as the lay term for relations.

One-to-One relationships

- The participation of both entities E_i and E_j in a relationship R_k is partial, i.e., E_i and E_j have min-max cardinalities $(0, 1)$ w.r.t. R. In this case, both, foreign key and relationship relation approaches can be applied to map R_k that preserves 3NF, depending on the choice of allowing or not null values in the foreign key.

  - *Posting a primary key*. The relationship R_k is accounted for by including the primary key of $\mathsf{rel}(E_j)$ as a foreign key with null values allowed in $\mathsf{rel}(E_i)$ (or the primary key of $\mathsf{rel}(E_i)$ as foreign key in $\mathsf{rel}(E_j)$), such that the conditions in C3 are satisfied. The following constraints must be added:
    - `UNIQUE` constraint on the foreign key fk_{i_j} of $\mathsf{rel}(E_i)$ (or on fk_{j_i} of $\mathsf{rel}(E_j)$).
  - *Define a separate relation*. The relationship R_k is represented by a separate relation $\mathsf{rel}(R_k)$, such that the conditions in C2 are satisfied. *One of the foreign keys* from FK_k is declared to act as a primary key of $\mathsf{rel}(R_k)$. The following constraints must be added:
    - On the foreign key of $\mathsf{rel}(R_k)$, that was not declared as a primary key, add `NOT NULL` and `UNIQUE` constraints.

Example 3.1.1. Consider a binary relationship Has between entities Lecturer and Office, as displayed in figure 3.1(a). Suppose a lecturer can have his/her own office or also can be a visiting lecturer, in which case he/she does not have an office. On the other hand, the office may or may not belong to a lecturer. Then there is a potential for null values in the foreign key O.Id in Lecturer relation, if we use a foreign key approach, as can be seen from the relation schemas on the left-hand side of figure 3.1(b). If the relative number of lecturers not having an office is large, and null values cannot be tolerated, then a better alternative is to set up a third relation Has, as displayed on the right-hand side of figure 3.1(b). Note that both relations, Lecturer and Has describe lecturers (as well as Office and Has describe offices). The former pertains to all lecturers, while the latter contains a



(a) Relationship with cardinalities (0, 1) and (0, 1)

Lecturer(L.SSN, L.Name, O.Id)
Office(L.SSN, L.Name)

Lecturer(L.SSN, L.Name)
Office(O.Id, O.Id)
Has(L.SSN, O.Number)

(b) Translation of a relationship

Figure 3.1: Mapping one-to-one relationship with cardinalities (0, 1) and (0, 1)

subset of all lecturers. Then we have a constraint that the set of lecturers' SSN in Has relation is always included in the set of lecturers' SSN in the Lecturer relation. More precisely, we have the following inclusion dependencies:

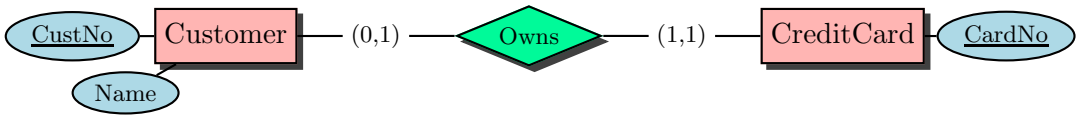
$$\mathsf{Has}\,[\mathsf{L.SSN}] \subseteq \mathsf{Lecturer}\,[\mathsf{L.SSN}] \text{ and } \mathsf{Has}\,[\mathsf{O.Id}] \subseteq \mathsf{Office}\,[\mathsf{O.Id}]\,.$$

Since every lecturer has at most one office and an office can belong to at most one lecturer, then if L.SSN is selected to act as a primary key of the relation Has, all values that occur in O.Number must be `NOT NULL` and `UNIQUE`. In the case of having only two relations, the values of the foreign key O.Id of Lecturer must be `UNIQUE` with nulls allowed (a lecturer does not necessarily have an office).

- The min-max cardinalities for one of the two entities, say E_i is (0, 1), and (1, 1) for E_j in the relationship R_k. In this case, the *foreign key approach* is applied, since null values do not occur in the foreign key (because of total participation). Thus, the primary key of the relation $\mathsf{rel}(E_i)$ is included as a foreign key in the relation $\mathsf{rel}(E_j)$, such that the conditions in C3 are satisfied. The following constraints must be added:
  - On the foreign key of fk_{j_i} of $\mathsf{rel}(E_j)$, add `NOT NULL` and `UNIQUE` constraints.

Example 3.1.2. Let Owns be a binary relationship between entities Customer and CreditCard, where Owns relates to the customers of a bank zero or one credit card. In figure 3.2(a), every credit card must belong to a customer, but a customer may not have a credit card. In this case we have two relations, as displayed by the two relation schemas in figure 3.2(b), with the foreign key attribute C.CustNo in the relation CreditCard). Since every credit card belongs to exactly one customer, the foreign key C.CustNo must have `NOT NULL` and `UNIQUE` values. Furthermore, the relation CreditCard contains a subset of all customers - only those that posses a credit card. Thus, we have the following inclusion dependency:

$$\mathsf{CreditCard}\,[\mathsf{C.CustNo}] \subseteq \mathsf{Customer}\,[\mathsf{C.CustNo}]\,.$$



(a) Relationship with cardinalities $(0, 1)$ and $(1, 1)$

Customer(C.CustNo, C.Name)
CreditCard(CC.CardNo, C.CustNo)

(b) Translation of a relationship

Figure 3.2: Mapping one-to-one relationship with cardinalities $(0, 1)$ and $(1, 1)$

- The participation of both entities E_i and E_j in R_k is total, i.e., E_i and E_j have min-max cardinalities $(1, 1)$. Again, since there is no potential for null values and in order to preserve the minimal number of relations in a relational schema, the foreign key approach is to be used. Thus, either include the primary key of $\mathsf{rel}(E_i)$ as a foreign key in $\mathsf{rel}(E_j)$ or include the primary key of $\mathsf{rel}(E_j)$ as a foreign key in $\mathsf{rel}(E_i)$, such that the conditions in C3 are satisfied. Add the following constraints (assume that the primary key of $\mathsf{rel}(E_i)$ is included in $\mathsf{rel}(E_j)$ as a foreign key):
  - On the foreign key of fk_{j_i} of $\mathsf{rel}(E_j)$, add `NOT NULL` and `UNIQUE` constraints.
  - All values that are in the primary key pk_i of $\mathsf{rel}(E_i)$ must coincide with the values in the foreign key fk_{j_i} of $\mathsf{rel}(E_j)$, i.e.,

$$\pi_{pk_i}(\mathsf{rel}(E_i)) - \pi_{fk_{j_i}}(\mathsf{rel}(E_j)) = \emptyset.$$

Example 3.1.3. Consider the ER diagram in figure 3.3(a). In this case, with a mapping we obtain two relations, and suppose the primary key M.SSN of Manager relation is posted as a foreign key to Project, as in figure 3.3(b). Since every manager manages exactly one project, then all managers from Manager relation must appear in the foreign key M.SSN of Project. That is,

$$\pi_{\mathsf{M.SSN}}(\mathsf{Manager}) - \pi_{\mathsf{M.SSN}}(\mathsf{Project}) = \emptyset.$$

Furthermore, from total participations of the corresponding entities we have `NOT NULL` and `UNIQUE` constraints on the foreign key M.SSN of Project relation.

One-to-Many relationships

- The min-max cardinality of one entity, say E_i is $(0, 1)$ and $(1, n)$ for the entity E_j in the relationship R_k. In this case, both, foreign key and relationship relation approaches can be applied, depending on the choice of allowing or not null values in the foreign key.
  - *Posting a primary key.* The relationship R_k is accounted for by including the primary key of $\mathsf{rel}(E_j)$ as a foreign key with null values allowed in $\mathsf{rel}(E_i)$,

(a) Relationship with cardinalities $(1, 1)$ and $(1, 1)$

Manager(M.SSN, M.Name, M.Salary)
Project(P.Code, P.Title, M.SSN)

(b) Translation of a relationship

Figure 3.3: Mapping one-to-one relationship with cardinalities $(1, 1)$ and $(1, 1)$

such that the conditions in C3 are satisfied. The following constraint must be added:

* All values that are in the primary key pk_j of $\mathsf{rel}(E_j)$ must coincide with the distinct values in the foreign key fk_{i_j} of $\mathsf{rel}(E_i)$, i.e.,

$$\pi_{pk_j}(\mathsf{rel}(E_j)) - \pi_{fk_{i_j}}(\mathsf{rel}(E_i)) = \emptyset.$$

– *Define a separate relation*. The relationship R_k is represented by a separate relation $\mathsf{rel}(R_k)$, such that the conditions in C2 are satisfied. Only *one of the foreign keys* of $\mathsf{rel}(R_k)$ forms its primary key, and namely fk_{k_i}. The following constraints must be added:

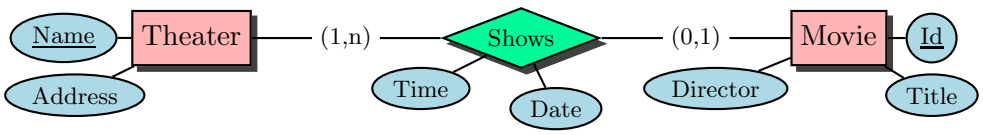
* All values over the primary key pk_j of $\mathsf{rel}(E_j)$ coincide with the distinct values of fk_{k_j} in $\mathsf{rel}(R_k)$, i.e.,

$$\pi_{pk_j}(\mathsf{rel}(E_j)) - \pi_{fk_{k_j}}(\mathsf{rel}(R_k)) = \emptyset.$$

* `NOT NULL` constraint on the foreign key fk_{k_j} of $\mathsf{rel}(R_k)$.

Note, that the π (`project`) operation *removes any duplicate tuples*, so the result of the `project` operation is a set of tuples, and hence a valid relation. On the other hand, if duplicates were not eliminated, the result would be a *multiset* or *bag* of tuples rather than a set. Although this is not allowed in the formal relational model, it is permitted in practice, e.g., SQL allows the user to specify whether duplicates should be eliminated or not. However, it is obvious that whenever we have $(1, n)$ cardinality constraint, we are possibly dealing with duplicate values in a relation. Thus, in these particular cases (i.e., whenever $(1, n)$ occurs on at least one side of the relationship), with the *equality* of projections we express that the values of a particular attribute in one relation coincide with the values of a particular attribute from another relation, i.e., we express the tuple equality IC.

Example 3.1.4. Consider the relationship Shows between entities Theater and Movie, as depicted in figure 3.4(a). Suppose movies may be displayed together



(a) Relationship with cardinalities $(0, 1)$ and $(1, n)$

Theater(T.Name, T.Address)
Movie(M.Id, M.Title, M.Director, T.Name, S.Time, S.Date)

(b) Translation of a relationship: two relations

Theater(T.Name, T.Address)
Movie(M.Id, M.Title, M.Director)
Shows(M.Id, T.Name, S.Time, S.Date)

(c) Translation of a relationship: three relations

Figure 3.4: Mapping one-to-many relationship with cardinalities $(0, 1)$ and $(1, n)$

with a theater in which they are shown, and also directly without the theater, in which case no information about show time and date is displayed. Then there is a potential for null values of T.Name, S.Time and S.Date in the Movie relation, if the mapping of 3.4(b) is applied. If the relative number of movies that are not being shown in any theater is large, and null values cannot be tolerated, a better alternative is to set up a third relation, as in 3.4(c). Note that both relations, Movie and Shows, describe movies. The former pertains to all movies. Shows relation contains a subset of all movies - those that are being shown in a particular theater. Thus we have the following constraint:

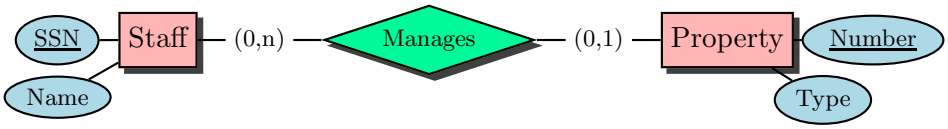
$$\mathsf{Shows}\,[\mathsf{M.Id}] \subseteq \mathsf{Movie}\,[M.Id]$$

On the other hand, $(1, n)$ on Theater entity side implies that every theater that appears in the Theater relations, must also occur in the Shows relation. Thus,

$$\pi_{\mathsf{T.Name}}(\mathsf{Theater}) - \pi_{\mathsf{T.Name}}(\mathsf{Movie}) = \emptyset.$$

Since the relation Shows describes the movies that are being shown in a theater, the values of T.Name in Shows must have `NOT NULL` values.

- The min-max cardinality of one entity, say E_i is $(0, 1)$ and $(0, n)$ for the entity E_j in the relationship R_k. Again, as already discussed in the previous case, both approaches can be applied.
  - *Posting a primary key*. Include the primary key of $\mathsf{rel}(E_j)$ as a foreign key with null values allowed in $\mathsf{rel}(E_i)$, such that the conditions in C3 are satisfied. No additional constraints are implied.
  - *Define a separate relation*. The relationship R_k is represented by a separate relation $\mathsf{rel}(R_k)$, such that the conditions in C2 are satisfied. *One of the*



(a) Relationship with cardinalities $(0, 1)$ and $(0, n)$

Staff(S.SSN, S.Name)
Property(P.Number, P.Type, S.SSN)

Staff(S.SSN, S.Name)
Property(P.Number, P.Type)
Manages(P.Number, S.SSN)

(b) Translation of a relationship

Figure 3.5: Mapping one-to-many relationship with cardinalities $(0, 1)$ and $(0, n)$

foreign keys forms the primary key of $\mathsf{rel}(R_k)$, and namely, fk_{k_i}. The following constraints must be added:

* `NOT NULL` constraint on the foreign key fk_{k_j}.

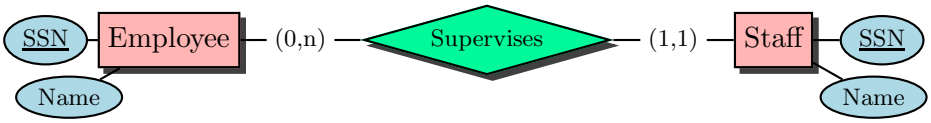
Example 3.1.5. Consider the ER diagram in figure 3.5(a). As already discussed before, if the number of such properties that are not managed by the stuff is relatively large, then instead of having two relations, it is better to set up three relations, as shown in figure 3.5(b). Then Manages relation contains the subset of all properties - those that are managed by the stuff, as well as the subset of all staff members - those that manage a property. Thus, we have the following inclusion dependencies, which, in fact, are implied by the definition of the foreign key:

$\mathsf{Manages}[\mathsf{P.Number}] \subseteq \mathsf{Property}[\mathsf{P.Number}]$ and $\mathsf{Manages}[\mathsf{S.SSN}] \subseteq \mathsf{Stuff}[\mathsf{S.SSN}]$.

Since all values of S.SSN that occur in Manages relation are only those staff members that manage a property, then these values must be restricted to be `NOT NULL`.

- The min-max cardinality of one entity, say E_i is $(1, 1)$ and $(0, n)$ for the entity E_j in the relationship R_k. In this case, the foreign key approach is to be applied, since the null values are not present, i.e., each instance of the entity E_i is related to exactly one instance of the entity E_j in the relationship R_k. Thus, include the primary key of the relation $\mathsf{rel}(E_j)$ as a foreign key in $\mathsf{rel}(E_i)$, such that the conditions in C3 are satisfied. Add the following constraints:
  - `NOT NULL` constraint on the foreign key fk_{i_j} of $\mathsf{rel}(E_i)$.

Example 3.1.6. Let Supervises be the relationship between entities Employee and Staff, as in figure 3.6(a). Because of $(1, 1)$ cardinality on the Staff side, the foreign key E.SSN of the relation Staff must have `NOT NULL` values, i.e., for each staff member, there is exactly one supervising employee, but each employee supervises several staff members or none of them. Therefore, we have that



(a) Relationship with cardinalities $(1, 1)$ and $(0, n)$

Employee(E.SSN, E.Name)
Staff(S.SSN, S.Name, E.SSN)

(b) Translation of a relationship

Figure 3.6: Mapping one-to-many relationship with cardinalities $(1, 1)$ and $(0, n)$

Staff relation contains a subset of all employees - only those that supervise stuff members, i.e.,

$$\mathsf{Staff}\,[\mathsf{E.SSN}] \subseteq \mathsf{Employee}\,[\mathsf{E.SSN}]\,.$$

- The min-max cardinality of one entity, say E_i is $(1, 1)$ and $(1, n)$ for the entity E_j in the relationship R_k. Here, for the same reason as in previous case, we choose the foreign key approach, and therefore include the primary key of the relation $\mathsf{rel}(E_j)$ as a foreign key in $\mathsf{rel}(E_i)$, such that the conditions in C3 are satisfied. Add the following constraints:
  - All values over the primary key pk_j of $\mathsf{rel}(E_j)$ coincide with the distinct values of fk_{i_j} in $\mathsf{rel}(E_i)$, i.e.,

$$\pi_{pk_j}(\mathsf{rel}(E_j)) - \pi_{fk_{i_j}}(\mathsf{rel}(E_i)) = \emptyset.$$

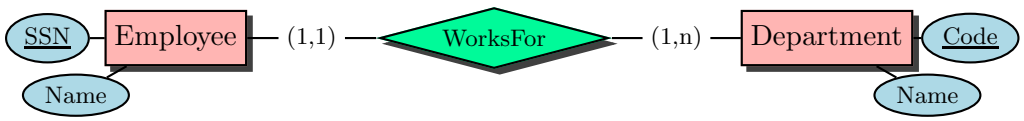
  - `NOT NULL` constraint on the foreign key fk_{i_j} of $\mathsf{rel}(E_i)$.

Example 3.1.7. As an example, consider the ER diagram in figure 3.7(a). Again, because of $(1, 1)$ cardinality of the entity Employee, we have `NOT NULL` constraint on the values of the foreign key D.Code in Employee relation, i.e., every employee works for exactly one department, but several employees may work in the same department. Additionally, $(1, n)$ on Department side enforces every value of the primary key D.Code of the relation Department also appear in the foreign key D.Code of Employee relation at least once. It implies that at any time, the projection on D.Code of the relation Employee must be equal to the projection on D.Code of Department relation, i.e.,

$$\pi_{\mathsf{D.Code}}(\mathsf{Department}) - \pi_{\mathsf{D.Code}}(\mathsf{Employee}) = \emptyset.$$

Many-to-Many relationships

As already noted before, for mapping many-to-many binary relationships to the relational model, the relationship relation approach must be applied. Therefore, for the



(a) Relationship with cardinalities $(1, 1)$ and $(1, n)$

Employee(E.SSN, E.Name, D.Code)
Department(D.Code, D.Name)

(b) Translation of a relationship

Figure 3.7: Mapping one-to-many relationship with cardinalities $(1, 1)$ and $(1, n)$

relationship R_k between two entities E_i and E_j with many-to-many cardinality constraints, a new relation $\mathsf{rel}(R_k)$ is created with the primary keys of $\mathsf{rel}(E_i)$ and $\mathsf{rel}(E_j)$ as foreign keys in $\mathsf{rel}(R_k)$, such that the conditions in C2 are satisfied. The primary key of $\mathsf{rel}(R_k)$ is the *combination of the primary keys* of $\mathsf{rel}(E_i)$ and $\mathsf{rel}(E_j)$ (trivially because of possibly repeating values in both foreign key attributes). As for additional constraints, we distinguish the three possible combinations of min-max cardinalities of each side of the relationship R_k, i.e., $(1, n)$ and $(0, n)$, $(1, n)$ and $(1, n)$, and $(0, n)$ and $(0, n)$ for the entities E_i and E_j, respectively.

- The min-max cardinality on both sides of the relationship R_k is partial, i.e., $(0, n)$ for the entities E_i and E_j. In this case, no additional constraints (except of inclusion dependencies following the foreign key definition) are implied.

  An explanatory example in provided below.

  Example 3.1.8. Consider the ER diagram in figure 3.8(a). Here, every employee may work on several projects or may be involved in none of them and the same with a project - there may be many employees working on it, or no employees at all (which in real world should not occur). Thus, the relation WorksOn describes the subset of all employees, as well as the subset of all projects. That is,

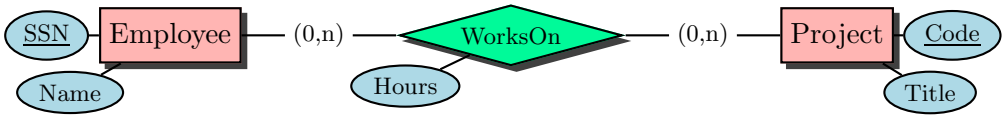
  $$\mathsf{WorksOn}\,[\mathsf{E.SSN}] \subseteq \mathsf{Employee}\,[\mathsf{E.SSN}] \text{ and } \mathsf{WorksOn}\,[\mathsf{P.Code}] \subseteq \mathsf{Project}\,[\mathsf{P.Code}]\,.$$

- The min-max cardinality of one entity, say E_i is $(0, n)$ and $(1, n)$ for the entity E_j in the relationship R_k. In this case, add the following constraints:

  – All values over the primary key of $\mathsf{rel}(E_j)$ coincide with the distinct values of fk_{k_j} in $\mathsf{rel}(R_k)$, i.e.,

  $$\pi_{pk_j}(\mathsf{rel}(E_j)) - \pi_{fk_{k_j}}(\mathsf{rel}(R_k)) = \emptyset.$$

  As an explanatory example, consider the ER diagram in figure 3.9(a), where the reasoning is based on those discussed previously.

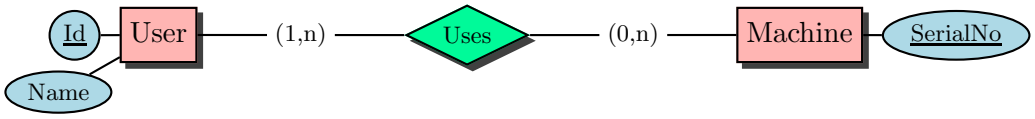


(a) Relationship with cardinalities $(0, n)$ and $(0, n)$

Employee(E.SSN, E.Name)
Project(P.Code, P.Title)
WorksOn(E.SSN, P.Code, W.Hours)

(b) Translation of a relationship

Figure 3.8: Mapping many-to-many relationship with cardinalities $(0, n)$ and $(0, n)$

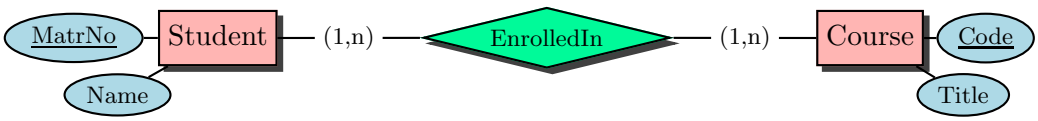


(a) Relationship with cardinalities $(0, n)$ and $(1, n)$

User(U.Id, U.Name)
Machine(M.SerialNo)
Uses(U.Id, M.SerialNo)

(b) Translation of the relationship

Figure 3.9: Mapping many-to-many relationship with cardinalities $(0, n)$ and $(1, n)$



(a) Relationship with cardinalities $(1, n)$ and $(1, n)$

Student(S.MatrNo, S.Name)
Course(C.Code, C.Title)
EnrolledIn(S.MatrNo, C.Code)

(b) Translation of the relationship

Figure 3.10: Mapping many-to-many relationship with cardinalities $(1, n)$ and $(1, n)$

- The min-max cardinality of both sides of the relationship R_k is total, i.e., $(1, n)$ for the entities E_i and E_j. In this case, add the following constraints:
  - All values over the primary key of $\mathsf{rel}(E_i)$ (resp. $\mathsf{rel}(E_j)$) coincide with the distinct values of fk_{k_i} (resp. fk_{k_j}) in $\mathsf{rel}(R_k)$, i.e.,

$$\pi_{pk_i}(\mathsf{rel}(E_i)) - \pi_{fk_{k_i}}(\mathsf{rel}(R_k)) = \emptyset \text{ and } \pi_{pk_j}(\mathsf{rel}(E_j)) - \pi_{fk_{k_j}}(\mathsf{rel}(R_k)) = \emptyset.$$

Example 3.1.9. Let EnrolledIn be the relationship between entities Student and Course with total participation, as displayed in figure 3.10(a). $(1, n)$ on Student side says that every student from Student relation must occur in EnrolledIn relation at least once. Thus at any time, the projection of Student relation on S.MatrNo is always equal to the projection of EnrolledIn on S.MatrNo. Analogous applies to Course and EnrolledIn pair of relations. Therefore, we have the following constraints:

$$\pi_{\mathsf{S.MatrNo}}(\mathsf{Student}) - \pi_{\mathsf{S.MatrNo}}(\mathsf{EnrolledIn}) = \emptyset,$$

$$\pi_{\mathsf{C.Code}}(\mathsf{Course}) - \pi_{\mathsf{C.Code}}(\mathsf{EnrolledIn}) = \emptyset.$$

3.1.4 Mapping Recursive Relationships

A recursive relationship R_k from an entity E_i to itself is modeled either applying foreign key or relationship relation approach. The option for choosing the appropriate approach depends on conditions discussed in section 3.1.3 (i.e., if null values are tolerated in order to reduce the number of relations in a relational schema, etc.). Thus, the same rules as for binary relationships can be applied to recursive relationships with corresponding cardinality constraints. In general, a single entity with a one-to-one relationship implies some form of entity occurrence pairing, as indicated by the relationship name. However, it is important to note that this pairing may be either completely optional (partial) or completely mandatory (total) [37], e.g., if in figure 3.11(a) a person is necessarily married ($(1, 1)$ cardinality in the relationship MarriedTo), then his/her spouse cannot be single (which is implied by cardinality

$(0,1)$). Many-to-many recursive relationships should also be designed as completely partial or completely mandatory. However, if for instance in [37] for one-to-many recursive relationships all combinations are allowed, we argue that the correct semantics can be represented only with cardinalities $(0,1)$ and $(0,n)$, or $(1,1)$ and $(0,n)$. In summary, the following six combinations are considered to be correct for representing recursive relationships:

- for a *one-to-one relationship*[4] either $(0,1)$ and $(0,1)$ or $(1,1)$ and $(1,1)$;
- for a *one-to-many relationship* either $(0,1)$ and $(0,n)$ or $(1,1)$ and $(0,n)$;
- for a *many-to-many relationship* either $(0,n)$ and $(0,n)$ or $(1,n)$ and $(1,n)$.

With a foreign key approach, the pairing entity key appears as a foreign key in the resulting table, as a reference to itself. That is, the foreign key attribute is taken from the same domain but is given different name to designate its unique use. Using a relationship relation approach, the relationship R_k is represented by a separate relation $\mathsf{rel}(R_k)$ including two foreign key attributes, corresponding to the primary key of $\mathsf{rel}(E_i)$ and their names in $\mathsf{rel}(R_k)$ represent the two roles of E_i in the relationship R_k. The many-to-many recursive relationship always results in a new relation. We list key conditions and additional constraints below, according to the cardinality of the two roles in a recursive relationship. The constraints are deduced with the same reasoning as for binary relationships.

- If a recursive relationship R_k is mapped by creating a separate relation $\mathsf{rel}(R_k)$ (i.e., with a relationship relation approach)[5], then
  - for a one-to-one relationship, only *one of the foreign keys* form the primary key of $\mathsf{rel}(R_k)$ (i.e., any of the two foreign keys of $\mathsf{rel}(R_k)$);
  - for a one-to-many relationship, only *one of the foreign keys* form the primary key of $\mathsf{rel}(R_k)$, and that foreign key represents the role with cardinality $(0,1)$ or $(1,1)$ in E_i;
  - for a many-to-many relationship, *both foreign keys* form the primary key of $\mathsf{rel}(R_k)$.
- For a *one-to-one* relationship mapped by applying foreign key approach
  - *optional* participation implies `UNIQUE` values on the foreign key of $\mathsf{rel}(E_i)$ with nulls allowed;
  - *mandatory* participation of a role A implies `NOT NULL` and `UNIQUE` values on the foreign key of $\mathsf{rel}(E_i)$ and a constraint

$$\pi_{pk_i}(\mathsf{rel}(E_i)) - \pi_{fk_{i_A}}(\mathsf{rel}(E_i)) = \emptyset,$$

    where fk_{i_A} represents role A of E_i in relationship R_k.

[4] With *relationship* in this subsection we refer to *recursive relationship*.

[5] If the relationship R_k is represented by posting the primary key of the relation $\mathsf{rel}(E_i)$ as a foreign key to itself, the primary key of $\mathsf{rel}(E_i)$ is not reflected.

- For a *one-to-one* relationship mapped by applying relationship relation approach, on the foreign key that was not declared as a primary key, add `NOT NULL` and `UNIQUE` constraints.
- For a *one-to-many* relationship mapped by applying foreign key approach *mandatory participation* on *one* side implies `NOT NULL` values in the foreign key of $\mathsf{rel}(E_i)$
- For a *one-to-many* relationship mapped by applying relationship relation approach (in fact this relationship has cardinalities $(0, 1)$ and $(0, n)$), on the foreign key that represents the role with *many* cardinality, add `NOT NULL` constraint.
- For a *many-to-many* relationship R_k, *mandatory* participation (i.e., $(1, n)$) of a role A implies constraint

$$\pi_{pk_i}(\mathsf{rel}(E_i)) - \pi_{fk_{k_{iA}}}(\mathsf{rel}(R_k)) = \emptyset,$$

  where $fk_{k_{iA}}$ represents role A of E_i in relationship R_k.

Example 3.1.10. In figure 3.11 an example for each, one-to-one, one-to-many and many-to-many recursive relationship is displayed, together with corresponding mappings to relational model. The diagram of 3.11(a) expresses that a person can be married to another person. Figure 3.11(b) says that every employee is managed by one of the other employees, and one employee can manage several other employees. As for the last diagram in figure 3.11(c), suppose that each employee has the opportunity to coauthor a report with one or more other employees, or to write a report alone[6].

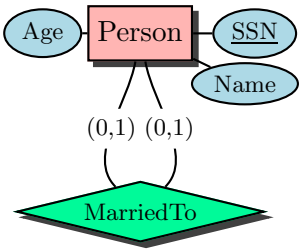
Consider the diagram in figure 3.11(b). According to the rules defined previously, with a mapping methodology we obtain a single relation Employee with a foreign key E.MngrSSN, which relates to every employee a manager (which is also employee) and references the same relation Employee. Since every employee has a manager, then the values of E.MngrSSN must be `NOT NULL`. On the other hand, the set of all managers defined in the foreign key E.MngrSSN is a subset of all employees. Thus, we have the following inclusion dependency:

$$\mathsf{Employee}\,[\mathsf{E.MngrSSN}] \subseteq \mathsf{Employee}\,[E.SSN]\,.$$

3.1.5 Mapping N-ary Relationships

N-ary relationships follow the same mapping rules as many-to-many relationships, i.e., uses the relationship relation approach. Thus, for each n-ary relationship R_k ($n > 2$), create a relation $\mathsf{rel}(R_k)$, such that the conditions in C2 are satisfied. However, the meaning of the keys depends on cardinalities of participating entities in a relationship. When all entities in an n-ary relationship R_k are *one*, the relation $\mathsf{rel}(R_k)$ consists of n possible distinct keys. That is, n functional dependencies are needed to describe this relationship. In general, the number of entities in R_k with connectivity *one* determines

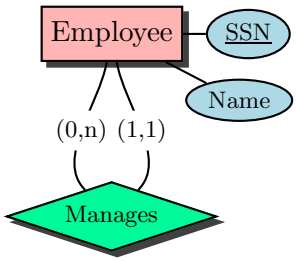
[6] This example was taken from [37].



Person(P.SSN, P.Name, P.SpouseSSN)

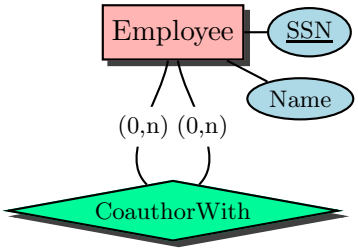
Person(P.SSN, P.Name)
MarriedTo(P.SSN, P.SpouseSSN)

(a) One-to-one, both sides optional



Employee(E.SSN, E.Name, E.MngrSSN)

(b) One-to-many, one side mandatory, many side optional



Employee(E.SSN, E.Name)
CoauthorWith(E.AuthorSSN, E.CoauthorSSN)

(c) Many-to-many, both sides optional

Figure 3.11: Mapping recursive relationships

the lower bound on the number of functional dependencies. When all entities in R_k are *many*, the *combination of the foreign keys* of $\mathsf{rel}(R_k)$ forms its primary key. To be precise, we list the key conditions for n-ary relationships involving entities with cardinality *one*, and additional constraints below.

- For every entity E_i participating in R_k with cardinality *one*, $\mathsf{rel}(R_k)$ is associated with the functional dependency $\mathsf{rel}(R_k) : (\mathsf{FK}_k - fk_{k_i}) \rightarrow fk_{k_i}$. Then the primary key of $\mathsf{rel}(R_k)$ consists of $pk_k = \mathsf{FK}_k - fk_{k_i}$, where FK_k is defined as in C2 of section 3.1.2, and fk_{k_i} must have `NOT NULL` values.
- For all entities $E_1, \ldots, E_n$ participating in R_k, if E_i $(1 \leq i \leq n)$ has cardinality $(1,1)$ or $(1,n)$, then

  $$\pi_{pk_i}(\mathsf{rel}(E_i)) - \pi_{fk_{k_i}}(\mathsf{rel}(R_k)) = \emptyset.$$

  In other words, all values that are in the primary key pk_i of $\mathsf{rel}(E_i)$ coincide with the distinct values of the foreign key fk_{k_i} of $\mathsf{rel}(R_k)$.

Example 3.1.11. This example is taken from [37]. Figure 3.12(a) shows the ternary one-to-one-to-one relationship Uses between entities Employee, Notebook and Project. The cardinalities says that an employee uses exactly one notebook for each project. Each notebook belongs to one employee for each project. Note that an employee may still work on many projects and use different notebooks for different projects. The relation Uses is associated with the following functional dependencies:

$$\mathsf{E.SSN}, \mathsf{P.Code} \rightarrow \mathsf{N.Id}, \ \mathsf{E.SSN}, \mathsf{N.Id} \rightarrow \mathsf{P.Code} \text{ and } \mathsf{P.Code}, \mathsf{N.Id} \rightarrow \mathsf{E.SSN}.$$

We choose $\mathsf{E.SSN}, \mathsf{P.Code}$ as a primary key of the relation Uses. Because of $(1,1)$ cardinalities on all participating entities we deduce the following constraints:

$$\pi_{\mathsf{E.SSN}}(\mathsf{Employee}) - \pi_{\mathsf{E.SSN}}(\mathsf{Uses}) = \emptyset,$$

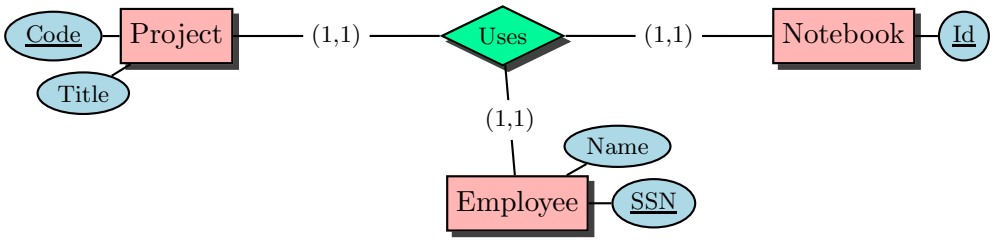
$$\pi_{\mathsf{P.Code}}(\mathsf{Project}) - \pi_{\mathsf{P.Code}}(\mathsf{Uses}) = \emptyset,$$

$$\pi_{\mathsf{N.Id}}(\mathsf{Notebook}) - \pi_{\mathsf{N.Id}}(\mathsf{Uses}) = \emptyset,$$

i.e., the projections on the primary keys of relations Employee, Project and Notebook are equal to the projections on the corresponding foreign keys of relation Uses.

3.1.6 "Missing" Cardinalities

Observe that we do not assume our ER schema to be completely settled with min-max cardinalities on every participating entity. We rather point out that whenever we have the case, where the cardinality of an entity in a relationship is not present ("missing"), we assume it to be $(0,n)$. Obviously, $(0,n)$ enforces the constraints that are also valid for $(0,1)$, $(1,1)$ and $(1,n)$. In fact, the only constraints determined by $(0,n)$ cardinality are inclusion dependencies. We call these constraints *guaranteed constraints*. Therefore $(0,n)$ is "placed" on the entity with "missing" cardinality and the corresponding mapping rule is applied.



(a) A ternary relationship with cardinalities (1, 1)

Employee(E.SSN, E.Name)
Project(P.Code, P.Title)
Notebook(N.Id)
Uses(E.SSN, P.Code, N.Id)

(b) Translation of the relationship

Figure 3.12: Mapping n-ary relationship with all cardinalities of (1, 1)

3.1.7 Mapping IS-A Relationships

Almost all textbooks (e.g., [17, 14]) describe several techniques for designing relational schemas in the presence of subclass hierarchies. The selection of an appropriate option is dependent on a number of factors such as the disjointness and participation constraints, whether the subclasses are involved in distinct relationships, and the number of participants in IS-A relationship:

a. Map each entity into a separate table. This approach requires two adjustments: First, subclasses must inherit identifying attributes from a single superclass, in order to be able to generate keys for their tables. Second, in the table created for an immediate subclass, its key should also be set to reference the primary key of the superclass, as a way of maintaining inclusion constraints dictated by the IS-A relationship.

b. Expand inheritance, so that all attributes of the superclass appear on all its subclasses. Then generate tables as usual for the subclasses, though not for the superclass itself. This approach is used only when the subclasses cover the superclass. The drawback of this method is that we cannot define foreign key constraints, since we do not have a relation for the superclass.

c. Some researchers also suggest a third possibility: "Collapse up" the information about subclasses into the table for the superclass. This approach is used only when the subclasses are disjoint.

In order to avoid ambiguities in the reverse process (i.e. in identifying the ER components from relational database), the methodology defined here deals only with the approach a. described above. Thus, more precisely,

C4 For a group of IS-A relationships, E_1 IS-A E_0, ..., E_m IS-A E_0, where each E_i is a subclass of the superclass E_0, $1 \leq i \leq m$, and E_0 is represented by relation $\mathsf{rel}(E_0)$ as defined in C1 of section 3.1.1, create a relation $\mathsf{rel}(E_i)(\mathsf{Att}_i)$ for each subclass E_i, together with inclusion dependencies $\mathsf{rel}(E_i)[fk_{i_0}] \subseteq \mathsf{rel}(E_0)[pk_0]$, and such that Att_i is the union of two disjoint set of attributes, X_i and FK_i, as defined below:

- X_i is in a one-to-one correspondence with the set of (local) attributes of E_i, where the correspondence is specified as in condition C1 in 3.1.1;
- FK_i is a set of foreign key attributes such that $\mathsf{FK}_i = fk_{i_0}$, fk_{i_0} is compatible with pk_0 of $\mathsf{rel}(E_0)$, and $fk_{i_0} = pk_i$.

Note that we do not determine here the foreign keys obtained from the participation of a subclass in a recursive or binary one-to-one or one-to-many relationship. These cases are captured in the condition C3 of section 3.1.2. In fact, this particular case will be discussed in more detail in the next chapter.

If there are constraints (disjoint or covering) on the subclasses $E_1, \ldots, E_m$ of the superclass E_0, then add the following exclusion and covering integrity constraints on corresponding relations[7].

- If subclasses $E_1, \ldots, E_m$ $(m \geq 2)$ are *disjoint*, then

$$\pi_{pk_1}(\mathsf{rel}(E_1)) \cap \ldots \cap \pi_{pk_m}(\mathsf{rel}(E_m)) = \emptyset,$$

  i.e., every tuple appearing in $\mathsf{rel}(E_i)$ must not appear in $\mathsf{rel}(E_j)$.

- If subclasses $E_1, \ldots, E_m$ $(m \geq 2)$ are *covering*, then

$$\pi_{pk_1}(\mathsf{rel}(E_1)) \cup \ldots \cup \pi_{pk_m}(\mathsf{rel}(E_m)) = \pi_{pk_0}(\mathsf{rel}(E_0)),$$

  i.e., the union of all tuples from $\mathsf{rel}(E_i)$ and $\mathsf{rel}(E_j)$ must form the tuples of $\mathsf{rel}(E_0)$.

Example 3.1.12. Consider the IS-A relationship depicted in figure 3.13(a), where every employee can be either full-time employed or part-time employed. After mapping we obtain the three relations with a common primary key E.SSN. Because of disjointness constraint we have

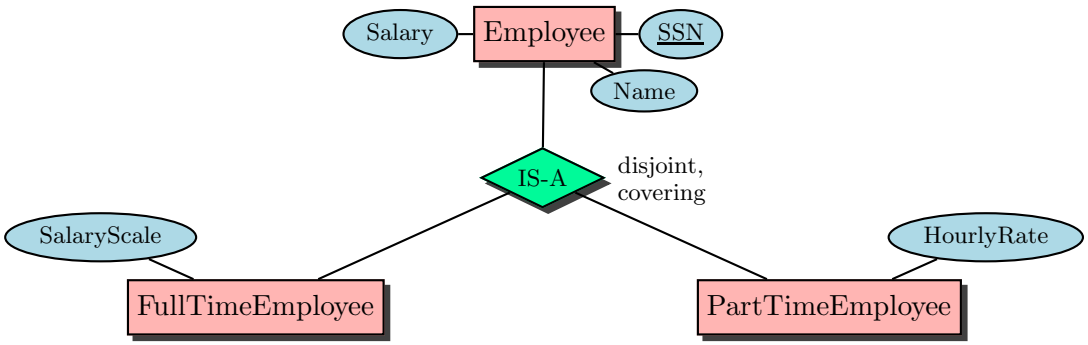
$$\pi_{\mathsf{E.SSN}}(\mathsf{FullTimeEmployee}) \cap \pi_{\mathsf{E.SSN}}(\mathsf{PartTimeEmployee}) = \emptyset,$$

that is, every employee which is full-time employed cannot be also part-time employed. From the covering constraint we obtain

$$\pi_{\mathsf{E.SSN}}(\mathsf{FullTimeEmployee}) \cup \pi_{\mathsf{E.SSN}}(\mathsf{PartTimeEmployee}) = \pi_{\mathsf{E.SSN}}(\mathsf{Employee}),$$

i.e., every employee can only be full-time employed or part-time employed.

[7]Note that in order to represent the disjoint or covering constraints on the subclasses in IS-A relationship, the number of subclasses must be $m \geq 2$.



(a) IS-A relationship

Employee(E.SSN, E.Name, E.Salary)
FullTimeEmployee(E.SSN, FT.SalaryScale)
PartTimeEmployee(E.SSN, PT.HourlyRate)

(b) Translation of IS-A relationship

Figure 3.13: IS-A relationship with *disjoint* and *covering* constraints

3.2 ER-to-Relational Mapping: Case Study

We proceed in this section with the case study of the previous sections. We apply the methodology defined in section 3.1 to generate the relational schema, as well as to restrict the allowed values of their attributes and to constraint the values of the foreign keys w.r.t. the corresponding primary keys (tuple equality ICs).

The conceptual schema for the case study is provided in appendix A in figure A.1. The mapping of the schema into a relational schema is quite straightforward. The translated schema is provided in figure A.2, appendix A, where, for the sake of readability, SQL syntax is used for formalizing relations by means of `create table` statements. Attribute integrity constraints (`NOT NULL`, `UNIQUE`) are also given. Tuple equality ICs on the occurrence of values in the foreign key w.r.t. the referenced primary key, as well as exclusion and covering ICs are formalized with SQL `create assertion` statements. Note that we do not list here inclusion dependencies, but they are implied automatically for each foreign key of the relation.

We start by translating all the entities to relations with their own identifiers, as described in section 3.1.1. As for binary relationships, we start with the single one-to-one relationship Sponsors, with partial participation of both participating entities. Suppose it is known that almost all departments sponsor one institute (i.e., null values can be tolerated), then it is accounted for by posting the primary key of the relation Institute as a foreign key to Department, restricted to have unique values (every institute can be sponsored by at most one department). The two one-to-many binary relationships WorksFor and AssignedTo are also modeled by posting the primary keys of the relations Department and Project, respectively, as foreign keys to the the relation Employee. The three remaining many-to-many binary relationships are immediately

mapped to new relations, Has, IsAuthorOf and IsCoauthorOf.

The recursive relationship Manages is accounted for by posting the primary key of Employee relation as a foreign key to itself, as discussed in section 3.1.4. The n-ary relationship Uses is mapped straightforwardly to new relation, and the combination of E.SSN, Pr.Code is selected to act as a primary key. Finally, the IS-A relationship is mapped as discussed in section 3.1.7, thus obtaining the relations Permanent and Consultant with the primary key of the superset relation Employee as their identifying key.

3.3 ER-to-Relational Mapping: General Results

To summarize, with the methodology defined in this chapter we obtain the relational schema of the form $\mathcal{R} = (\mathbf{R}, \mathcal{I} \cup \mathcal{E} \cup \mathcal{A})$, where $\mathcal{I}$ denotes the set of inclusion dependencies, $\mathcal{E}$ denotes the set of tuple equality, exclusion and covering integrity constraints, and $\mathcal{A}$ denotes a set of attribute integrity constraints, and in particular, `NOT NULL` and `UNIQUE` constraints. The relations in $\mathbf{R}$ represent ER components, the inclusion dependencies of $\mathcal{I}$ represent the interaction of ER components, the constraints of $\mathcal{E}$ represent the occurrence of tuples in one relation w.r.t. other relation(s), and the constraints of $\mathcal{A}$ represent restrictions on the allowed values for the attributes of ER components.

In terms of ER schemas, all the cases analyzed in the methodology above imply that a single relation may correspond to the following ER components:

- an entity E_i, provided that this entity is not involved in any recursive or binary one-to-one or one-to-many relationship R_k, such that E_i participates in R_k with cardinality *one*;
- a subclass entity E_i, provided that this entity is not involved in any recursive or binary one-to-one or one-to-many relationship R_k, such that E_i participates in R_k with cardinality *one*;
- a recursive, binary or n-ary relationship R_k of any type, i.e. one-to-one, one-to-many, many-to-many;
- an entity E_i and one or several recursive or binary one-to-one or one-to-many relationships R_k, provided that E_i is involved in (each) R_k with cardinality *one*.

The following proposition characterizes the translation of an ER schema to relational schema. From now on, we denote by $\mathsf{rel}(R_i)$ the relation of $\mathbf{R}$ that may correspond to any component of ER schema.

Proposition 3.3.1. *Let $\mathcal{R} = (\mathbf{R}, \mathcal{I} \cup \mathcal{E} \cup \mathcal{A})$ be a relational schema obtained from the ER-to-Relational methodology described above. Let relation $\mathsf{rel}(R_i)(\mathsf{Att}_i)$ of $\mathbf{R}$ correspond to ER component C_i (either entity or relationship), and FK_i be the union of all foreign keys, fk_{i_j}, associated with $\mathsf{rel}(R_i)$, such that each fk_{i_j} references relation $\mathsf{rel}(R_j)$. Then every foreign key fk_{i_j} satisfies the following conditions:*

- *fk_{i_j} is either included in, or disjoint with the primary key of $\mathsf{rel}(R_i)$;*

- *fk_{i_j} is involved in an inclusion dependency* $\mathsf{rel}(R_i)[fk_{i_j}] \subseteq \mathsf{rel}(R_j)[pk_j]$ *corresponding to the interaction of C_i with another component C_j such that*
  - *C_i is an entity participating in a binary or recursive one-to-one or one-to-many relationship with another entity C_j (resp. with itself in case of recursive relationship), such that C_i has cardinality one, i $pk_i \cap \mathsf{FK}_i = \emptyset$;*
  - *C_i is a relationship involving C_j as a participating entity, i $pk_i \subseteq \mathsf{FK}_i$;*
  - *C_i is a subclass of C_j in IS-A relationship i $fk_{i_j} = \mathsf{FK}_i = pk_i$*[8].

Proof. The proof follows the specification of the methodology defined in section 3.1. □

3.4 Eliminating Relations

The relations obtained by the ER-to-Relational methodology preserve 3NF and are without *redundant* attributes, where an attribute is considered redundant, if its removal has no effect on the information capacity of the relation. It is worth to note that our methodology minimizes the number of relations in a mapping process. That is, particular relationships in ER schema are mapped by foreign key approach, instead of mapping every ER construct to a separate table. For this reason, the *merging* process of relations, suggested as a second step of transforming the ER schema to relational schema, is not relevant (e.g., as in [33]). However, in this section we identify the additional conditions under which the particular relations could be *eliminated*, in order to decrease the number of relations in a database. This relation eliminating process preserves the information capacity and 3NF of the relations over $\mathcal{R}$, as assured with the following definition.

Definition 3.4.1. Given a relational schema, $\mathcal{R} = (\mathbf{R}, \mathcal{I} \cup \mathcal{E} \cup \mathcal{A})$, and a relation from $\mathbf{R}$, $\mathsf{rel}(R_i)(\mathsf{Att}_i)$, such that

- $|\mathsf{Att}_i| = 1$ and $\mathsf{Att}_i = pk_i$;
- $\mathsf{rel}(R_i)$ corresponds to the entity with cardinality $(1, 1)$ or $(1, n)$ in ER schema;
- there exists a relation $\mathsf{rel}(R_j)$, such that $\mathsf{rel}(R_j)[fk_{j_i}] \subseteq \mathsf{rel}(R_i)[pk_i]$;
- $\mathsf{rel}(R_i)$ is not involved in the right-hand side of any other inclusion dependency of $\mathcal{I}$ of the form: $\mathsf{rel}(R_k)[fk_{k_i}] \subseteq \mathsf{rel}(R_i)[pk_i]$, $k \neq i$, $\mathsf{rel}(R_k) \in \mathbf{R}$.

Then $\mathsf{rel}(R_i)$ can be eliminated from $\mathbf{R}$.

After elimination of such relations $\mathsf{rel}(R_i)$ from $\mathcal{R} = (\mathbf{R}, \mathcal{I} \cup \mathcal{E} \cup \mathcal{A})$, the resulting schema $\mathcal{R}' = (\mathbf{R'}, \mathcal{I}' \cup \mathcal{E}' \cup \mathcal{A}')$ is obtained as follows:

- $\mathbf{R}'$ results by eliminating $\mathsf{rel}(R_i)$ from $\mathbf{R}$;

[8] For more discussion on this case, refer to section 4.3.3 of chapter 4.

- $\mathcal{I}'$ results by eliminating from $\mathcal{I}$ the inclusion dependency

$$\mathsf{rel}(R_j)\,[fk_{j_i}] \subseteq \mathsf{rel}(R_i)\,[pk_i]\,,$$

  and $\mathsf{rel}(R_j)$ is defined as in definition 3.4.1;

- $\mathcal{E}'$ results by eliminating from $\mathcal{E}$ the constraint

$$\pi_{pk_i}(\mathsf{rel}(R_i)) - \pi_{fk_{j_i}}(\mathsf{rel}(R_j)),$$

  and $\mathsf{rel}(R_j)$ is defined as in definition 3.4.1;

- $\mathcal{A}'$ is equal to $\mathcal{A}$, since there are no constraints from $\mathcal{A}$ on $\mathsf{rel}(R_i)$.

Note that the elimination process cannot be seen as relation merging process. Intuitively, the relation can be eliminated if it, first of all, is related to any relation (a relationship is expressed with inclusion dependency). More precisely, it has to be subsumed by another relation, but in such a way that every tuple from one relation is also included in another relation. Therefore, the total participation ($(1,1)$ or $(1,n)$) for the relation being eliminated is required. Observe that if the relation having several attributes is eliminated, the information about the objects associated with this relation is lost. Therefore, the requirement for having the single primary key attribute is issued. Finally, we do not want to eliminate the relation, whose primary key acts as a foreign key in another relation, in order not to lose information on relationships between objects.

An example of applying the elimination technique is given below.

Example 3.4.1. As a simple example, consider the diagram in figure A.1 from appendix A. It can be easily seen from the list of relations in figure A.2 that the relation Notebook satisfies all conditions in definition 3.4.1. That is, Notebook has the only attributes which is its primary key and corresponds to the entity with cardinality $(1,1)$, which means that all notebooks described by Notebook relation, are also contained in the Uses relation. Moreover, there exists an inclusion dependency

$$\mathsf{Uses}\,[\mathsf{N.Id}] \subseteq \mathsf{Notebook}\,[\mathsf{N.Id}]\,,$$

and, obviously, Notebook relation is not involved in any other inclusion dependency. Thus, Notebook relation can be eliminated, without losing any information.

Finally, we would like to notice that eliminating relations in a relational database, whose schema is the translation of an ER schema, usually obscures the semantics of the database application for users. While the meaning of a relation in a relational schema without eliminated relations remains comprehensible enough, the semantics of a relation after elimination process is usually ambiguous. We therefore will try to identify all ambiguities in the next chapter.

Chapter 4

Extracting the Conceptual Schema with a Set of Views

In this chapter we investigate the problem of extracting the set of views from relational schema, where each view corresponds to a single ER construct (instead of corresponding to several ER constructs). We consider relational schemas involving key-based inclusion dependencies, together with a set of constraints on attributes and occurring values.

The methodology described in previous chapter for translating the conceptual data model into a relational data model is one of the sources used for carrying out the views construction from an existing relational database. Since the design principles provided there are generally accepted as leading to a good relational model given an ER model, reverse structure extraction proceeds by analyzing the primary and foreign keys of relations and relationships between them, and then asking "what kind of ER component would give rise to this?". The best answer to this question comes from chapter 3 by looking at each ER component which could give rise to the relational structure, and selecting the most likely alternative. However, to extract as much semantics as possible from a relational schema, only key analysis technique is not sufficient. Thus, we also build an analysis of constraints defined explicitly in referential integrity constraints and implicitly in data sources.

4.1 Assumptions Regarding the Input Database

In order to be able to make guarantees, we have to limit ourselves to "standard" relational schemas, since otherwise it is impossible to derive the intended meaning of an arbitrary table. For this reason, let us consider only relational schemas generated by the ER-to-Relational mapping methodology, as described in section 3.1 of chapter 3. Thus, we have a relational database that preserves the 3NF and the information about the data schema with relation names, attribute names, primary and foreign keys, as well as data instances with additional constraints are available.

Furthermore, we assume that inclusion dependencies in our relational schema do not contain cycles. In other words, the ER diagram, from which the relational schema

was generated, does not have cycles.

Regarding the naming of relations, we assume that the relations have globally unique names in a relational schema. As for attributes of the relations, primary key attributes and foreign key attributes in different relations having identical names (which happens typically) are renamed, whenever necessary, in a way that removes the ambiguity. After the renaming, the referential constraints should become obvious. Based on this, we assume that attributes with the same name have an identical domain and the same interpretation whenever they occur.

4.2 Classification of Relations

Based on the results of ER-to-Relational mapping methodology, described in section 3.3, and more precisely, based on the appearance of primary and foreign keys in a relation, we classify the relations into the four categories.

- We call a relation $\mathsf{rel}(R_i)$, having primary key pk_i disjoint with every of its foreign key fk_{i_j} (if any), *base relation*. That is, $pk_i \cap \mathsf{FK}_i = \emptyset$. From section 3.3 follows that a base relation corresponds to one of the following ER components: i) an entity, or ii) an entity and one or several recursive or binary one-to-one or one-to-many relationships, provided that this entity is involved in (each) relationship with cardinality *one*.

- A relation $\mathsf{rel}(R_i)$, whose primary key pk_i is included in the set of its foreign keys FK_i, is called *relationship relation*. That is, $pk_i \subseteq \mathsf{FK}_i$. Such relations, according to section 3.3, correspond to one of the following ER components: i) a recursive relationship of any type (i.e., one-to-one, one-to-many, many-to-many), ii) binary relationship of any type, or iii) n-ary relationship of any type.

- A relation $\mathsf{rel}(R_i)$, whose primary key pk_i is equal to its *single* foreign key fk_{i_j}, is called *specific relation*, i.e., $fk_{i_j} = \mathsf{FK}_i = pk_i$. Relation of this type correspond to the subclass entity in IS-A relationship in a conceptual schema, provided that this entity is not involved in any recursive or binary one-to-one or one-to-many relationship, such that the subclass entity participates in this relationship with cardinality *one*.

- A relation $\mathsf{rel}(R_i)$, that does not satisfy any of conditions above, is treated as *ambiguous relation*.

4.3 The Extraction Process

We have shown in section 3.1 that conceptual schema can be mapped into relational schema $\mathcal{R}$ of the form $\mathcal{R} = (\mathbf{R}, \mathcal{I} \cup \mathcal{E} \cup \mathcal{A})$, where $\mathbf{R}$, $\mathcal{I}$, $\mathcal{E}$ and $\mathcal{A}$ denote set of relations in a relational schema, set of inclusion dependencies, set of tuple equality, exclusion and covering integrity constraints, and set of `NOT NULL` and `UNIQUE` constraints on attributes, respectively, and where every $\mathsf{rel}(R_i)$ of $\mathbf{R}$ corresponds to one or several components of the conceptual schema. The properties of such relations were captured

in proposition 3.3.1, and are used below for the extraction of views and identification of ER components.

4.3.1 Setting up the Notation

We summarize the notation used in this chapter for attributes of relations, that follow one from the previous chapter. For the relation $\mathsf{rel}(R_i)(\mathsf{Att}_i)$ of $\mathbf{R}$, Att is the set of attributes of $\mathsf{rel}(R_i)$, comprising of the primary key pk_i, the set of non-key attributes X_i and the set of foreign key attributes FK_i, where fk_{i_j} of FK_i references the primary key pk_j of $\mathsf{rel}(R_j)$ (in other words, there is an inclusion dependency in $\mathcal{I}$ $\mathsf{rel}(R_i)[fk_{i_j}] \subseteq \mathsf{rel}(R_j)[pk_j]$). Thus, $\mathsf{Att}_i = \{pk_i, \mathsf{X}_i, \mathsf{FK}_i\}$.

Given a relational schema $\mathcal{R} = (\mathbf{R}, \mathcal{I} \cup \mathcal{E} \cup \mathcal{A})$, we construct in the next section the set of view definitions $\mathcal{V} = \{V_1, \ldots, V_n\}$ over the same schema $\mathcal{R}$, such that each view corresponds to a unique ER component. Additionally, for every relation $\mathsf{rel}(R_i)$, if it represents

- an entity in a conceptual schema, then we identify an entity[1] E_i:`entity`[2];
- a relationship in a conceptual schema, then we identify a relationship[3] $R_{1,\ldots,n}(E_1, \ldots, E_n)$:`relationship` between entities $E_1, \ldots, E_n$;
- an IS-A relationship in a conceptual schema, then we identify a relationship $E_i(E_0)$:`subclass`, where E_i denotes a subclass and E_0 its superclass in IS-A relationship.

For each identified relationship $R_{1,\ldots,n}(E_1, \ldots, E_n)$, to each E_i $(1 \leq i \leq n)$ we associate a *cardinality*, that defines its cardinality constraints in a relationship R_k. We denote the cardinality of E_i by $\mathsf{card}(E_i) = (k, l)$, for every $i \in \{1, \ldots, n\}$, and $k, l \in \{0, 1, n\}$.

For a set of identified IS-A relationships $E_1(E_0)$:`subclass`, ..., $E_m(E_0)$:`subclass`, such that $m \geq 2$, we associate to the set of subclasses $E_1, \ldots, E_m$ a *constraint*, that defines the disjoint and/or covering constraints between subclasses in IS-A relationship. We denote the constraints between subclasses $E_1, \ldots, E_m$, by $\mathsf{constr}(E_1, \ldots, E_m) = \mathit{disjoint}$ and/or $\mathsf{constr}(E_1, \ldots, E_m) = \mathit{covering}$, thus expressing the disjointness and covering constraints, respectively.

4.3.2 Constructing Views and Identifying ER Components

First of all, the set of relations $\mathbf{R}$ in the schema $\mathcal{R} = (\mathbf{R}, \mathcal{I} \cup \mathcal{E} \cup \mathcal{A})$ are classified to the four categories as described in section 4.2. The interaction of ER components is determined by inclusion dependency characterization in proposition 3.3.1. Then, for every category of relations, under certain conditions construct the views as follows:

[1] For the name of the entity (and IS-A relationships defined below) we use the name of relation.

[2] Every E_i $(1 \leq i \leq n)$ in this and the following sections simply denotes the name of the entity, which in fact is the same as the one used for corresponding relation.

[3] We assume that the user gives a meaningful name for the identified relationship. For the sake of clarity, in this thesis we use the names from running examples.

- For every *base relation* $\mathsf{rel}(R_i)$ in a relational schema $\mathcal{R}$, such that $\mathsf{rel}(R_i)$ does not contain any foreign key attributes ($\mathsf{rel}(R_i)$ is not involved on the left-hand side of any inclusion dependency), i.e., $\mathsf{Att}_i = \{pk_i, \mathsf{X}_i\}$,
  - create a view $V_i = \pi_{\mathsf{Att}_i}(\mathsf{rel}(R_i))$;
  - identify an entity E_i:`entity`.
- For every *base relation* $\mathsf{rel}(R_i)$ in a relational schema $\mathcal{R}$, such that the set of foreign keys FK_i is not empty,
  - create a view $V_i = \pi_{pk_i,\mathsf{X}_i}(\mathsf{rel}(R_i))$[4];
  - identify an entity E_i:`entity`;
  - for every inclusion dependency of $\mathcal{I}$, where $\mathsf{rel}(R_i)$ appears on the left-hand side, i.e., $\mathsf{rel}(R_i)[fk_{i_j}] \subseteq \mathsf{rel}(R_j)[pk_j]$, create a view $V_{i,j} = \pi_{pk_j,pk_i}(\mathsf{rel}(R_j) \bowtie \mathsf{rel}(R_i))$;
  - identify a binary relationship $R_{i,j}(E_i, E_j)$:`relationship`[5], for every such inclusion dependency in $\mathcal{I}$ where $\mathsf{rel}(R_i)$ appears on the left-hand side.
- For every *relationship relation* $\mathsf{rel}(R_i)$ in a relational schema $\mathcal{R}$,
  - create a view $V_i = \pi_{\mathsf{Att}_i}(\mathsf{rel}(R_i))$;
  - identify a relationship $R_{1,\ldots,n}(E_1, \ldots, E_n)$:`relationship`, where every E_i $(1 \leq i \leq n)$ corresponds to each of the relations $\mathsf{rel}(R_j)$, such that $\mathsf{rel}(R_i)[fk_{i_j}] \subseteq \mathsf{rel}(R_j)[pk_j]$, n - number of such inclusion dependencies of $\mathcal{I}$.
- For every *specific relation* $\mathsf{rel}(R_i)$ in a relational schema $\mathcal{R}$,
  - create a view $V_i = \pi_{pk_i,\mathsf{X}_i}(\mathsf{rel}(R_i))$;
  - identify an IS-A relationship $E_i(E_0)$:`subclass`, where E_0 corresponds to the relation $\mathsf{rel}(R_j)$, such that $\mathsf{rel}(R_i)[fk_{i_j}] \subseteq \mathsf{rel}(R_j)[pk_j]$.

Example 4.3.1. As an example, consider the set of tables in figure A.2 from the previous chapter. According to the conditions in section 4.2, the following relations are classified as base relations: Employee, Department, Project, Notebook, Institute, Researcher, Publication - every foreign key of these relations is disjoint with their primary key. The relations Uses, Has, IsAuthorOf and IsCoauthorOf are classified as relationship relations, since the primary key of every relation is included in the set of its foreign key. Finally, as the primary key ESSN of the relations Permanent and Consultant coincide with the only foreign key, these relations are classified as specific relations.

Let's look at how views can be generated and ER components identified for each type of the relation, according to the procedure defined above. The view for the base

[4]We are unable to identify the attributes, belonging to the relationship in ER diagram. For now, all non-key attributes in a base relation are treated as belonging to the entity. In general, we expect the user to verify attributes assigned to the views.

[5]As already discussed before, base relation represents only a recursive or binary one-to-one or one-to-many relationship between entities E_i and E_j, and E_i has cardinality *one*.

relation that does not have any foreign key attributes is generated straightforwardly. For instance, for the base relation Institute, we define the view

$$V_{\mathsf{Institute}} = \pi_{\mathsf{InstId,Name}}(\mathsf{Institute}),$$

and identify an entity Institute:`entity`. Let's take the base relation Employee. Again, firstly we define a view $V_{\mathsf{Employee}} = \pi_{\mathsf{ESSN,Name,BDate,StartTime,Hours}}(\mathsf{Employee})$, and we identify an entity Employee:`entity`. Furthermore, every foreign key in the relation identifies a relationship. Thus, we define the following views and identify the corresponding relationships:

$V_{\mathsf{Employee,Employee}} = \pi_{\mathsf{MngrSSN,ESSN}}(\mathsf{Employee} \bowtie \mathsf{Employee})$,
Manages(Employee, Employee):`relationship`;

$V_{\mathsf{Employee,Department}} = \pi_{\mathsf{DeptId,ESSN}}(\mathsf{Department} \bowtie \mathsf{Employee})$,
WorksFor(Employee, Department):`relationship`;

$V_{\mathsf{Employee,Project}} = \pi_{\mathsf{Code,ESSN}}(\mathsf{Project} \bowtie \mathsf{Employee})$,
AssignedTo(Employee, Project):`relationship`.

Consider the relationship relation Uses. Following the procedure above, we define the view $V_{\mathsf{Uses}} = \pi_{\mathsf{ESSN,Code,Id}}(\mathsf{Uses})$. Since three foreign keys are defined for Uses relation, we identify the ternary relationship Uses(Employee, Project, Notebook):`relationship`.

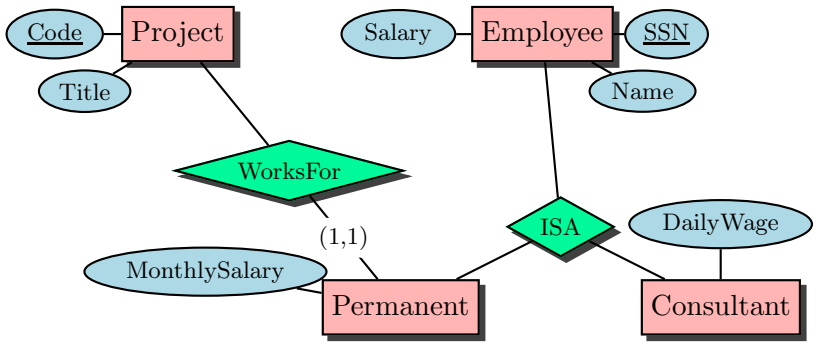
Finally, let's look at the specific relation Permanent. According to the extraction rules we have the view $V_{\mathsf{Permanent}} = \pi_{\mathsf{MonthlySalary}}(\mathsf{Permanent})$, and IS-A relationship Permanent(Employee):`subclass`.

4.3.3 Dealing with Ambiguities

It is worth to note that the characterization of keys correlation for the relations representing subclasses of IS-A relationship covers only the case, when the subclass does not participate in other relationship. Namely, in proposition 3.3.1, that is the core for identifying the ER constructs from the relational database, we declare that for the relation $\mathsf{rel}(R_i)$, corresponding to the subclass E_i of IS-A relationship, the *only* foreign key fk_{i_j} of $\mathsf{rel}(R_i)$ is equal to its primary key pk_i. This is always the case, when the subclass E_i is not involved in any recursive or binary one-to-one or one-to-many relationship, such that E_i participates in this relationship with cardinality *one*. Thus, whenever for a relation $\mathsf{rel}(R_i)$, $fk_{i_j} = \mathsf{FK}_i = pk_i$, we can *guarantee* that $\mathsf{rel}(R_i)$ corresponds to the subclass of IS-A relationship.

However, the use of inheritance together with embedded foreign key in a relation for representing the IS-A relationship and a recursive or binary relationship, make the relation ambiguous, in the sense that this relation cannot be classified correctly, according to the rules in section 4.2. The relevant example is provided below.

Example 4.3.2. Consider a modified part of the study case diagram, depicted in figure 4.1(a). In particular, we are interested in the relation Permanent, corresponding to the subset entity in ER diagram. Observe that the primary key E.SSN of Permanent relation is included in the set of its foreign keys {E.SSN, Pr.Code}. Following the classification defined in section 4.2, this relation is classified as a relationship relation, which in fact should be classified as a specific relation.



(a) An example ER diagram

Employee(E.SSN, E.Name, E.Salary)
Project(Pr.Code, Pr.Title)
Permanent(E.SSN, P.SalaryScale, P.Code)
Consultant(E.SSN, C.DailyWage)

(b) Translating the ER diagram

Figure 4.1: An example of ambiguity with subclass in IS-A relationship

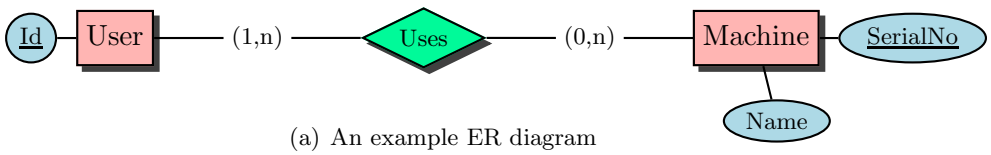
Other ambiguities arise in a schema with eliminated relations. Recall that one of the conditions for the relation, $\mathsf{rel}(R_i)$, being eliminated is that $\mathsf{rel}(R_i)$ must be referenced by another relation, $\mathsf{rel}(R_j)$, i.e., there must be inclusion dependency $\mathsf{rel}(R_j)[fk_{j_i}] \subseteq \mathsf{rel}(R_i)[pk_i]$. Clearly, after the relation $\mathsf{rel}(R_i)$ has been eliminated, the inclusion dependency is also eliminated. In other words, fk_{j_i} does not act anymore as a foreign key of $\mathsf{rel}(R_j)$. Thus, most of the times, the relation $\mathsf{rel}(R_j)$ is classified incorrectly.

Example 4.3.3. Consider the ER diagram in figure 4.2(a). The relation User satisfies all conditions to be eliminated. Thus, the relations after User relation has been eliminated are listed in figure 4.2(b). Now the relation Uses has the combination of U.Id, M.SerialNo as its primary key, and SerialNo acts as its foreign key (note that U.Id is not the foreign key). We see that none of the conditions defined in classification on relations is satisfied. The relation is considered as being ambiguous.

In general, we expect the cases, where an entity has the only attribute (the condition for the relation to be eliminated) to be quite rare though (e.g., [38] considers that ER diagram having entity with a single attribute has to be redesigned).

In fact, it has been verified that the relations from the relational schema $\mathcal{R}'$ (where relations have been eliminated), can be classified as:

- base relations, *without ambiguity*;
- specific relations, *with ambiguity*;
- ambiguous relations, clearly *with ambiguity*.



(a) An example ER diagram

Machine(M.SerialNo, M.Name)
Uses(U.Id, M.SerialNo)

(b) Remaining relations after elimination

Figure 4.2: An example of ambiguity with eliminated relations

We have also examined that relations treated as ambiguous (i.e., that do not satisfy any of conditions for the three classes), are of the type as depicted in figure 4.2(a). Thus, the user is suggested to treat the relation, say $\mathsf{rel}(R_i)$, where its primary key pk_i is a proper superset of the set of its foreign keys FK_i (i.e., $\mathsf{FK}_i \subset pk_i$), as relationship relation, additionally establishing the connection with the entity, corresponding to the part of the primary key that is not the foreign key. For instance, in example above, Uses relation should be treated as relationship relation, connecting two entities, where the connection with the entity Machine is obvious. We have to establish a connection with another entity, thus constructing the views: $V_{\mathsf{User}} = \pi_{\mathsf{Id}}(\mathsf{User})$, with identified entity User:`entity`; and $V_{\mathsf{Uses}} = \pi_{\mathsf{U.Id,M.SerialNo}}(\mathsf{Uses})$, with identified relationship Uses(User, Machine):`relationship`.

For these consequences, we choose to deal with some of the ambiguity by relying on database designers (users), in the process of views extraction, instead of disallowing the subset entities of IS-A relationships to participate in binary or recursive relationships, or assuming that in ER diagram we do not have entities with a single attribute. Specifically, the user is supposed to verify the ER constructs identified together with every view for *specific relations*, and to assign the meaning for *ambiguous relations* (possibly following the suggestions described above).

4.3.4 Deciding on Cardinalities

In this section we present our approach to determine the cardinalities of the relationships in the conceptual schema. It is based only on the constraints defined for the particular relations, as analyzed in chapter 3. We do not query the data instances themselves (as it is done in [3], for instance), since the cardinalities obtained in this way cannot be guaranteed. That is, the fact that the number of tuples on selecting only the distinct values is equal to the number of tuples without restricting on distinct values does not necessarily determine the upper limit of the cardinality to be 1 (i.e., it may be still allowed to be n).

To provide an intuition, consider the relation $\mathsf{rel}(R_i)$, classified as base relation and having the foreign key fk_{i_j} that references relation $\mathsf{rel}(R_j)$. As already noted before, base relations having a foreign key represent only binary or recursive one-to-one or one-to-many relationships. Then if fk_{i_j} of $\mathsf{rel}(R_i)$ is constrained to be `NOT NULL`, then

the cardinality on the side of the entity corresponding to $\mathsf{rel}(R_i)$ is $(1,1)$ (i.e., every instance of the entity corresponding to $\mathsf{rel}(R_i)$ participates in a relationship represented by the same relation $\mathsf{rel}(R_i)$ at least once). On the other hand, if the foreign key fk_{i_j} is constrained to be `UNIQUE`, then the cardinality of the entity corresponding to the relation $\mathsf{rel}(R_j)$ may be either $(1,1)$ or $(0,1)$ (i.e., every instance of the entity corresponding to $\mathsf{rel}(R_j)$ participates in a relationship represented by relation $\mathsf{rel}(R_i)$ at most once). Then to decide on mandatory participation, we check if there is a constraint on projections, etc.

Note that the tuple equality integrity constraint in analysis of the previous chapter was expressed in terms of relational algebra (i.e., having the form $\pi_{fk_{i_j}}(\mathsf{rel}(R_i)) - \pi_{pk_j}(\mathsf{rel}(R_j)) = \emptyset$). However, we cannot determine exactly how it should be implemented in a chosen DBMS (i.e., using `check` clause or `create assertion` statement, etc). As a consequence, we provide a general algorithm, where this particular constraint is referred to as tuple equality IC.

Following the observations above, we construct the algorithm for the relations classified as base relations that decides on cardinality constraints for participating entities within a relationship, which is displayed in Algorithm 1. It takes as input a database $\mathcal{D}$ over a relational schema $\mathcal{R}$ such that the relations over $\mathbf{R}$ are classified as base relations or relationship relations. The algorithm returns the cardinalities determined for corresponding entities participating in a relationship that given $\mathsf{rel}(R_i)$ represents.

As for relationship relations, we first of all distinguish the cases where the foreign key of a relation coincides with its primary key, and where the foreign key is either included in or disjoint with the primary key of a relation. The case when a foreign key of a relation is equal to its primary key indicates that the relationship is either one-to-one or one-to-many. Clearly, the foreign key that coincides with the primary key references the relation which corresponds to the entity with cardinality *one*. The following step is to verify if this foreign key participates in a tuple equality integrity constraint that actually represents mandatory participation in a relationship. The intuition behind for other steps follows that in previous case for base relations. A detailed algorithm is provided in Algorithm 2.

Lastly, we consider specific relationship relations that correspond to IS-A relationships in a conceptual schema. The disjointness and covering constraints (if they exist) are extracted immediately by checking if the primary keys of the relations corresponding to subclasses of IS-A participate in a particular constraint. Specifically, we take the primary keys, $pk_1, \ldots, pk_m$ of a set of specific relations $\mathsf{rel}(R_1), \ldots, \mathsf{rel}(R_m)$ $(m \geq 2)$, corresponding to entities $E_1, \ldots, E_m$, respectively, that have the same superclass, E_0, and verify if there is a constraint $\pi_{pk_1} \cap \ldots \cap \pi_{pk_m} = \emptyset$ and/or constraint $\pi_{pk_1} \cup \ldots \cup \pi_{pk_m} = \pi_{pk_0}$. The former indicates the subclasses to be disjoint and we refer to it in the Algorithm 3 as exclusion integrity constraint, while the latter indicates the subclasses to be covering, and we refer to it as covering integrity constraint. This is done for generality, as already discussed before. That is, we cannot determine in which form these constraints are represented in an underlying DBMS (in our example we use SQL `create assertion` statements).

The verifying algorithm is provided in Algorithm 3. It takes the set of relations,

Algorithm 1: Find cardinalities for entities participating in the relationships represented by base relations.

Input: A database $\mathcal{D}$ over a relational schema $\mathcal{R}$ such that relations over $\mathbf{R}$ are classified as base relations.

Output: Cardinalities for corresponding entities participating in a relationship.

Let $fk_{i_1}, \ldots, fk_{i_n}$ be a nonempty set of f.k.'s of $\mathsf{rel}(R_i)$, referencing relations $\mathsf{rel}(R_1), \ldots, \mathsf{rel}(R_n)$ resp.;
foreach *base relation* $\mathsf{rel}(R_i)$ **do**
  foreach *f.k.* fk_{i_j} *of* $\mathsf{rel}(R_i)$ **do**
    /* Decide on cardinalities of E_i entity side, corresponding to $\mathsf{rel}(R_i)$ */;
    if fk_{i_j} *is NOT NULL* **then**
      $\mathsf{rel}(R_i)$ represents an entity E_i with cardinality $\mathsf{card}(E_i) = (1, 1)$;
    else
      $\mathsf{rel}(R_i)$ represents an entity E_i with cardinality $\mathsf{card}(E_i) = (0, 1)$;
    end
    /* Decide on cardinalities of E_j entity side, corresponding to $\mathsf{rel}(R_j)$ */;
    if fk_{i_j} *is UNIQUE* **then** /* the cardinality of E_j is *one* */
      if fk_{i_j} *participates in a tuple equality IC* **then** /* determines mandatory participation */
        $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (1, 1)$;
      else
        $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (0, 1)$;
      end
    /* otherwise, the cardinality of E_j is *many* */;
    else if fk_{i_j} *participates in a tuple equality IC* **then**
      $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (1, n)$;
      else
        $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (0, n)$;
      end
    end
  end
end

Algorithm 2: Find cardinalities for entities participating in the relationships represented by relationship relations.

Input: A database $\mathcal{D}$ over a relational schema $\mathcal{R}$ and a set of relations of $\mathbf{R}$ classified as relationship relations.

Output: Cardinalities for corresponding entities participating in a relationship.

Let $fk_{i_1}, \ldots, fk_{i_n}$ be a nonempty set of f.k.'s of $\mathsf{rel}(R_i)$, referencing relations $\mathsf{rel}(R_1), \ldots, \mathsf{rel}(R_n)$ resp. Let pk_i be p.k. of $\mathsf{rel}(R_i)$;
foreach *relationship relation* $\mathsf{rel}(R_i)$ **do**
  foreach *f.k.* fk_{i_j} *of* $\mathsf{rel}(R_i)$ **do**
    if $fk_{i_j} = pk_i$ **then** /* the cardinality of E_j is *one* */
      if fk_{i_j} *participates in a tuple euqlity IC* **then**
        $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (1, 1)$;
      else
        $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (0, 1)$;
      end
    if fk_{i_j} *is* `UNIQUE` **then** /* the cardinality of E_j is *one* */
      if fk_{i_j} *participates in a tuple equality IC* **then**
        $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (1, 1)$;
      else
        $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (0, 1)$;
      end
    /* otherwise, the cardinality of E_j is *many* */;
    else if fk_{i_j} *participates in a tuple equality IC* **then**
      $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (1, n)$;
      else
        $\mathsf{rel}(R_j)$ represents an entity E_j with cardinality $\mathsf{card}(E_j) = (0, n)$;
      end
    end
  end
end

classified as specific relationship relations $\mathsf{rel}(R_1)$, ..., $\mathsf{rel}(R_m)$, and a base relation $\mathsf{rel}(R_j)$ corresponding to the superclass, grouped by the same superclass. The algorithm returns the constraints (if any) satisfied by a set of subclasses in an IS-A relationship.

Algorithm 3: Find constraints on subclasses of IS-A relationships represented by specific relationship relations.

Input: A database $\mathcal{D}$ over a relational schema $\mathcal{R}$ and a set of relations classified as specific relations, grouped by base relation corresponding to the superclass.

Output: The constraints on subclasses in IS-A relationship.

Let $\mathsf{rel}(R_1), \ldots, \mathsf{rel}(R_m)$ be a group of specific relations, each of them corresponding to the subclasses $E_1, \ldots, E_m$, where $m \geq 2$ and let $\mathsf{rel}(R_0)$ be a base relation corresponding to the superclass E_0 of IS-A relationship. Let $pk_1, \ldots, pk_m$ and pk_0 be their corresponding primary keys;

foreach *group of relations* $\mathsf{rel}(R_1), \ldots, \mathsf{rel}(R_m), \mathsf{rel}(R_0)$ **do**
- `disjlist` := {};
- `coverlist` := {};
- **foreach** pk_i *of* $\mathsf{rel}(R_i)$, $1 \leq i \leq m$ **do**
  - **if** pk_i *participates in an exclusion IC* **then**
    - `/* add` E_i `corresponding to` $\mathsf{rel}(R_i)$ `to the disjoint subclasses list */`;
    - `disjlist` := `disjlist` $\cup \{E_i\}$;
  - **end**
  - **if** pk_i *participates in a covering IC* **then**
    - `/* add` E_i `corresponding to` $\mathsf{rel}(R_i)$ `to the covering subclasses list */`;
    - `coverlist` := `coverlist` $\cup \{E_i\}$;
  - **end**
- **end**
- $\mathsf{constr}($`disjlist`$) = \mathit{disjoint}$;
- $\mathsf{constr}($`coverlist`$) = \mathit{covering}$;

end

Chapter 5

Answering Queries over Conceptual Schema

In the extraction process of the previous chapter, together with a set of views, we elicited the data semantics of the relational database, in the sense that every view corresponds to the element in ER schema, and identified the ER constructs. However, the data model itself was not expressed (we did not form the ER graph). Thus, in this chapter we formalize the elicited ontology with Description Logics (DLs) [6] and show the process of answering queries over relational databases using their conceptual schema.

DLs are logics specifically suited for the representation of structured knowledge, and they provide the formal foundation for the ontology languages that are now becoming standard. Description Logics have been devised as formalizations of Frame-based systems [20] and Semantic Networks [35, 8, 4], to overcome the problems of these formalisms related to the lack of a clear formal semantics. In DLs, the domain of interest is structured through concepts, denoting sets of objects, and roles, denoting binary relations over the domain. Complex concept and role expressions are formed using specific constructs, and it is the set of allowed constructs that characterize a certain DL. DL knowledge bases are typically constituted by an intensional component (called TBox), asserting inclusions between concepts (and possibly roles), and an extensional component (called ABox), asserting membership of individuals in concepts and of pairs of individuals in roles.

In this chapter we first present the developed query answering system that enables users to pose queries over the conceptual schema of a database. As discussed before, such a system provides added value against conventional DBMSs, where the users are exposed the relational schema only. At the core of our work there is an idea of query answering by rewriting. In general, query answering by rewriting is divided in two phases. The first one re-expresses a user query posed over the conceptual schema in terms of the relations at the underlying database, and the second evaluates the rewriting over the underlying database.

To formalize the conceptual schema of the database, we extend the DL $DL-Lite$ [9], thus obtaining $DLR-Lite$, which is specifically tailored to capture conceptual

data models (e.g., Entity-Relationship), Object-oriented formalisms (e.g., UML class diagrams), basic ontology languages, while keeping low complexity of reasoning, in particular, polynomial in the size of the instances in the knowledge base.

We have devised a flexible way of mapping the conceptual level to the underlying relational level, which provides the users an SQL-like query language over the conceptual schema. Queries at the conceptual level are first translated into the relational level queries by taking into account the mapping of entities and relationships to the actual database relations. To provide a complete answer to the query, the system then uses the developed query rewriting technique to take into account the constraints expressed in the conceptual schema. The initial user query is thus translated to a set of SQL queries that are evaluated by the DBMS.

This rewriting technique adds value to conventional query answering techniques. Firstly, the user is allowed to formulate more simple queries using terms defined in the conceptual schema only, without taking into account some relational database related details (e.g., join attributes). Moreover, the query rewriting technique allows one to infer additional information that was not stated explicitly in the user query but is implied by the constraints at the conceptual level. Last but not least, the formalization of the conceptual schema and the developed reasoning technique allow checking the consistency of the underlying database against the conceptual schema, therefore, the trustiness of the information system is improved.

At the next step we adopt the query answering technique to the setting analyzed and defined in this thesis. That is, the relations defined in a query answering system can be described by means of views instead of being actual tables. We show all the process with a case study at the end of this chapter.

5.1 Formal Framework

Query answering system is a triple $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, where $\mathcal{K}$ is the knowledge base (KB) of $\mathcal{S}$, $\mathcal{R}$ is a relational schema for $\mathcal{S}$ and $\mathcal{M}$ is the mapping between the KB of $\mathcal{S}$ and the relational schema of $\mathcal{S}$.

5.1.1 Conceptual Level

We call our description logic language *DLR-Lite*, that allows to represent the domain of interest in terms of concepts, denoting sets of objects, and relationships, denoting relations between objects. In the language, *basic concepts* are defined as follows:

$$B \ ::= \ A \mid \exists[i]R$$

where A denotes an atomic concept, R an n-ary relationship, and $1 \leq i \leq n$. Intuitively, $\exists[i]R$ denotes the projection of R on the *i-th* component. Note, that all concepts denote unary predicates.

The correspondences between basic concepts and ER schema constructs are straightforward. Each entity in ER schema is represented by an atomic concept in *DLR-Lite*, and each relationship in ER schema by a relationship in *DLR-Lite*. The intuition behind a component is the ER-role identifier within a relationship in an ER diagram.

For representing intensional knowledge in the KB, we have assertions of the form:

$$B_1 \sqsubseteq B_2 \qquad \textit{(inclusion)}$$

$$B_1 \texttt{ disj } B_2 \qquad \textit{(disjointness)}$$

$$(\texttt{funct } \exists[i]R) \qquad \textit{(functionality)}$$

An inclusion assertion expresses that a basic concept is subsumed by another concept, a disjointness assertion states that the set of objects denoted by a basic concept B_1 is disjoint from the ones denoted by another concept B_2, while a functionality assertion expresses the (global) functionality of a certain component of a relationship.

The formal meaning of concept descriptions above is given in terms of interpretations over a fixed infinite countable *domain* Δ. We assume, we have one constant for each object, denoting exactly that object.

An *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ consists of a first order structure over Δ with an *interpretation function* $\cdot^{\mathcal{I}}$ such that:

$$A^{\mathcal{I}} \subseteq \Delta$$

$$R^{\mathcal{I}} \subseteq \Delta^n$$

$$(\exists[i]R)^{\mathcal{I}} = \{\ c \mid \exists(c_1, \ldots, c_n) \in R^{\mathcal{I}},\ c = c_i\}.$$

An interpretation $\mathcal{I}$ *satisfies* an inclusion assertion $B_1 \sqsubseteq B_2$ iff $B_1^{\mathcal{I}} \subseteq B_2^{\mathcal{I}}$; $\mathcal{I}$ satisfies a disjointness assertion $B_1 \texttt{ disj } B_2$ iff $B_1^{\mathcal{I}} \cap B_2^{\mathcal{I}} = \emptyset$; $\mathcal{I}$ satisfies a functionality assertion $(\texttt{funct } \exists[i]R)$ if $(c_1, \ldots, c_i, \ldots c_n) \in R^{\mathcal{I}} \wedge (c'_1, \ldots, c_i, \ldots, c'_n) \in R^{\mathcal{I}} \supset c_1 = c'_1, \ldots, c_n = c'_n$.

A *model of a KB* $\mathcal{K}$ is an interpretation $\mathcal{I}$ that satisfies all the assertions in $\mathcal{K}$. A KB is *satisfiable*, if it has at least one model. A KB $\mathcal{K}$ *logically implies* an assertion α if all the models of $\mathcal{K}$ satisfy α.

All presented assertions allow us to specify the typical constructs used in conceptual modeling, e.g., ER diagram. Specifically:

- *IS-A*, using assertions of the form $B_1 \sqsubseteq B_2$, stating that the class B_1 is a subclass of the class B_2;
- *class disjointness*, using assertions of the form $B_1 \texttt{ disj } B_2$, stating disjointness between the two classes B_1 and B_2[1];
- *role-typing*, using assertions of the form $\exists[i]R \sqsubseteq B$, stating that the *i-th* component of the relationship R is of type B;
- *participation constraints*, using assertions of the form $B \sqsubseteq \exists[i]R$, stating that instances of class B participate to the relationship R as the *i-th* component;
- *non-participation constraints*, using assertions of the form $B \texttt{ disj } \exists[i]R$, stating that instances of class B do not participate to the relationship R as the *i-th* component;
- *functionality restrictions*, using assertions of the form $(\texttt{funct } \exists[i]R)$, stating that an object can be the *i-th* component of the relationship R at most once.

[1] However, *DLR-Lite* in incapable to represent the *covering* constraint between classes

5.1.2 Relational Level

At the relational level we consider *relations* over relational schema $\mathcal{R}$, where each relation has an associated sequence of typed attributes. Each relation may have a sequence of one or more *components*, where each component is a sequence of attributes of the relation. Components may not overlap. We call attributes that do not belong to any component, *additional attributes* of the relation. Note, that the order of components and the order of attributes may not necessarily be related to each other.

5.1.3 Mapping from Conceptual to Relational Level

We can now define the mapping $\mathcal{M}$ between conceptual and logical level as follows:

- to each atomic concept A, $\mathcal{M}$ associates a relation $\mathcal{M}(A)$ with a single component;
- to each n-ary relationship R, $\mathcal{M}$ associates a relation $\mathcal{M}(R)$ with n components.

The mapping induces a *signature* on basic concepts, and specifically

- for an atomic concept A, the signature is the sequence of types of attributes of the component of the relation corresponding to A.
- for a concept of the form $\exists[i]R$, the signature is the sequence of types of the i-th component of the relation corresponding to R.

A mapping $\mathcal{M}$ is *consistent* with the conceptual level $\mathcal{K}$ and the relational level $\mathcal{R}$ of a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, if for each inclusion assertion $B_1 \sqsubseteq B_2$ in $\mathcal{K}$, the signature of B_1 is equal to the signature of B_2. Note that for disjointness assertions B_1 `disj` B_2, we do not require B_1 and B_2 to have the same signature. Indeed, if B_1 and B_2 have different signatures, the disjointness assertions will trivially be satisfied at the relational level. In the following, we will always assume that in a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, the mapping $\mathcal{M}$ is consistent with $\mathcal{K}$ and $\mathcal{R}$.

5.1.4 Semantics of a System $\mathcal{S}$

In order to define the semantics of a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, we first extend the mapping $\mathcal{M}$ to a mapping $\mathcal{M}_c$ from basic concepts to components of relations as follows:

- for an atomic concept A, let $\mathcal{A}$ be the sequence of attributes corresponding to the only component of $\mathcal{M}(A)$. Then $\mathcal{M}_c(A) = \pi_{\mathcal{A}}(\mathcal{M}(A))$;
- for a relationship R, let $\mathcal{A}$ be the sequence of attributes corresponding to the i-th component of $\mathcal{M}(R)$. Then $\mathcal{M}_c(\exists[i]R) = \pi_{\mathcal{A}}(\mathcal{M}(R))$.

A database $\mathcal{D}$ satisfies w.r.t. $\mathcal{S}$

- an inclusion assertion $B_1 \sqsubseteq B_2$, if $(\mathcal{M}_c(B_1))^{\mathcal{D}} \subseteq (\mathcal{M}_c(B_2))^{\mathcal{D}}$;

- a disjointness assertion B_1 `disj` B_2, if $(\mathcal{M}_c(B_1))^{\mathcal{D}} \cap (\mathcal{M}_c(B_2))^{\mathcal{D}} = \emptyset$
- a functionality assertion (`funct` $\exists[i]R$), if the cardinality of $(\mathcal{M}_c(\exists[i]R))^{\mathcal{D}}$ is equal to the cardinality of $(\mathcal{M}(R))^{\mathcal{D}}$. In other words, the set of attributes of the i-th component of R is a key of $R^{\mathcal{D}}$.

A database $\mathcal{D}$ is said to be *consistent* w.r.t. a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, if it satisfies w.r.t. $\mathcal{S}$ all assertions in $\mathcal{K}$. A database $\mathcal{D}$ is said to be *df-consistent* w.r.t. $\mathcal{S}$, if it satisfies w.r.t. $\mathcal{S}$ all disjointness and functionality assertions in $\mathcal{K}$.

5.2 Queries

5.2.1 Queries over Conceptual Level

Queries over a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$ are specified using an SQL-like syntax corresponding to *Select-Project-Join* (SPJ) queries. We exploit the availability of components of the relations to extend the standard SQL syntax with a new construct for the simplified specification of *join* conditions. Specifically, we enable the use of *component comparison expressions*. This extended semantics can be used to specify joins between components of the relations which correspond to the "abstract" definitions in the schema. More precisely, such a query is written in the form:

`SELECT` ⟨*attribute_specifications*⟩
`FROM` ⟨*relationship_specifications*⟩
`WHERE` ⟨*selection_conditions*⟩

where

- ⟨*relationship_specifications*⟩ denotes the concepts and relationships involved in the query and the way they join together. It is defined as follows:

  ⟨*relationship_specifications*⟩ ::= ⟨*rel_spec*⟩ | ⟨*relationship_specifications*⟩, ⟨*rel_spec*⟩
  ⟨*rel_spec*⟩ ::= ⟨*join*⟩ `ON` ⟨*conditions*⟩
  ⟨*join*⟩ ::= ⟨*relationship*⟩ | ⟨*join*⟩ `JOIN` ⟨*relationship*⟩
  ⟨*relationship*⟩ ::= C_i `AS` V_i
  ⟨*conditions*⟩ ::= ⟨*equality*⟩ | ⟨*conditions*⟩ `AND` ⟨*equality*⟩
  ⟨*equality*⟩ ::= $e_i = e_j$

  Intuitively, ⟨*relationship_specifications*⟩ is a sequence of expressions of one of the following forms:

  - C `AS` V
  - C_1 `AS` V_1 `JOIN` C_2 `AS` V_2 `JOIN` $\cdots$ `JOIN` C_k `AS` V_k `ON` $e_1 = e_2$ `AND` $\cdots$ `AND` $e_{h-1} = e_h$

  where

  - each C_j denotes the name of a relationship or an atomic concept in $\mathcal{K}$;

  - each V_j is a unique variable name, associated to C_j[2];
  - in the equalities $e_i = e_j$, each e_i or e_j is either
    * V, if V is a variable corresponding to an atomic concept;
    * $V.i$, if V is a variable corresponding to a relationship of arity $n \geq i$,
  - the signatures of the two associated concepts/relationships components must be the same.

- $\langle$*attribute_specifications*$\rangle$ is a sequence of attributes of the form $V.a$, where V is a variable in $\langle$*relationship_specifications*$\rangle$, associated to concept or relationship C, and a is an attribute of relation $\mathcal{M}(C)$;

- $\langle$*selection_conditions*$\rangle$ is a set of equalities, each of one of the following forms:

  - $V_1.a_1 = V_2.a_2$,
  - $V_1.a_1 = c$,

  where V_i is a variables in $\langle$*relationship_specifications*$\rangle$, associated to concept or relationships C, a_i is an attribute of relation $\mathcal{M}(C)$, and c is a constant.

5.2.2 Conjunctive Queries over the Relational Level

Since the query rewriting algorithm makes use of conjunctive queries, in this section we present how a conjunctive query over the relational level can be obtained from a query over the conceptual level.

Given a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, the conversion of a query q over the conceptual level into a conjunctive query over the relational level is done in two steps:

1. the query q is converted into a standard SQL SPJ query q' over the relational schema $\mathcal{R}$;

2. q' is converted into a conjunctive query using the standard translation.

In this conversion, the order of attributes of a relation R, specified at the relational level, is preserved in the atoms for R.

In order to convert our conceptual queries to standard SQL queries, first each relationship C_j is substituted with $\mathcal{M}(C_j)$. For each equality $e_1 = e_2$ in the conceptual query, we substitute it with the conjunction of equalities between the attributes corresponding to the components mentioned in e_1 and e_2.

As for the second step, it is well known that SQL SPJ queries can be translated into equivalent conjunctive queries [2]. Moreover, since we can translate conceptual queries with extended SQL syntax into standard SQL queries, in the process of query rewriting, without loss of generality, we will concentrate on queries formulated as CQs.

Given a query q over $\mathcal{S}$, we denote with $CQ(q, \mathcal{S})$ the conjunctive query over $\mathcal{R}$ resulting from the above conversion.

[2]Note that relationships and atomic concepts may be repeated.

In order to evaluate, using a relational DBMS, the queries we get from the rewriting procedure, we need to convert them back to SQL. In doing so, we again make use of the order of attributes specified at the relational level. We denote the conversion of a CQ q to SQL with $\mathsf{SQL}(q, \mathcal{R})$.

5.2.3 Reasoning in system $\mathcal{S}$

Given a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, a conceptual query q over $\mathcal{S}$ and a database $\mathcal{D}$ over $\mathcal{R}$, the *certain answers* $\mathsf{ans}(q, \mathcal{S}, \mathcal{D})$ is the set of tuples c of constants of Δ, such that $c \in q_{\mathcal{S}}^{\mathcal{D}'}$ for every database $\mathcal{D}'$ that includes $\mathcal{D}$ and is consistent with $\mathcal{S}$.

The basic reasoning services over a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$ are:

- *KB satisfiability*: verify whether a KB is satisfiable.
- *query answering*: given a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, a conceptual query q over $\mathcal{S}$ and a database $\mathcal{D}$ over $\mathcal{R}$, return the certain answers $\mathsf{ans}(q, \mathcal{S}, \mathcal{D})$.
- *query rewriting*: given a system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, and a conceptual query q over $\mathcal{S}$, return a query q_r over $\mathcal{R}$, such that $q_r^{\mathcal{D}} = \mathsf{ans}(q, \mathcal{S}, \mathcal{D})$ for every database $\mathcal{D}$ that is df-consistent with $\texttt{Normalize}(\mathcal{S})$.

5.3 Query Rewriting in System $\mathcal{S}$

In this section we present an algorithm that computes the perfect rewriting of a UCQ. Before proceeding, we address some preliminary issues.

df-consistency of $\mathcal{D}$ w.r.t. $\mathcal{S}$ The algorithm `Consistent` takes as input a normalized KB $\mathcal{K}$ and verifies the following conditions:

- if there exists a disjunction assertion B_1 `disj` B_2, such that $(\mathcal{M}_c(B_1))^{\mathcal{D}} \cap (\mathcal{M}_c(B_2))^{\mathcal{D}} \neq \emptyset$
- if there exists a functionality assertion (`funct` $\exists[i]R$), such that the cardinality of $(\mathcal{M}_c(\exists[i]R))^{\mathcal{D}}$ is not equal to the cardinality of $(\mathcal{M}(R))^{\mathcal{D}}$.

Informally, the first condition corresponds to checking whether $\mathcal{D}$ explicitly contradicts some disjunction assertion in $\mathcal{K}$, and the second condition corresponds to check whether $\mathcal{D}$ violates some functionality assertion in $\mathcal{K}$. If at least one of the above conditions holds, then the algorithm returns *false*, i.e., $\mathcal{D}$ is not fk-consistent w.r.t. $\mathcal{S}$. Otherwise, the algorithm returns *true*.

5.3.1 Rewriting

The basic idea of the method used is to reformulate the query taking into account the KB $\mathcal{K}$ [28]: in particular, given a query q over the conceptual schema $\mathcal{K}$, we compile the assertions of the KB into the query itself, thus obtaining a new query q'. Such a new query is then evaluated over the database instance $\mathcal{D}$.

We say that an argument of an atom in a query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body, or a constant, while we say that it is *unbound* if it corresponds to a non-distinguished non-shared variable (as usual, we use the symbol _ to represent non-distinguished non-shared variables). Notice that an atom of the form $\exists[i]R(x)$ has the same meaning as $R(\ldots, x, \ldots)$, where x occurs on the i-th position. For ease of exposition, in the following we will use the latter form.

Definition 5.3.1. We indicate with $gr(g, I)$ the atom obtained from the atom g by applying the inclusion assertion I as follows. An inclusion assertion $B \sqsubseteq A$ (resp. $B \sqsubseteq \exists[i]R$) is applicable to an atom $T(x_1, \ldots, x_n)$ if

(i) $\mathcal{M}(A) = T$ (resp. $\mathcal{M}(R) = T$)

(ii) all variables among $x_1, \ldots, x_n$ that are in positions of T that are not part of the only (resp. the $i-th$) component of T are unbound.

For $g = T(x_1, \ldots, x_n)$, $gr(g, A_1 \sqsubseteq A_2)$ (resp. $gr(g, A \sqsubseteq \exists[i]R)$; $gr(g, \exists[i]R \sqsubseteq A)$) is the atom $T'(x'_1, \ldots, x'_n)$, where

- $T' = \mathcal{M}(A_1)$, $T = \mathcal{M}(A_2)$ (resp. $T' = \mathcal{M}(A)$, $T = \mathcal{M}(R)$; $T' = \mathcal{M}(R)$, $T = \mathcal{M}(A)$);
- the variables in $T'(x'_1, \ldots, x'_n)$ that correspond to the only (resp. i-th) component of T' are equal to the ones that correspond to the only (resp. i-th) component of T;
- the remaining variables in $T'(x'_1, \ldots, x'_n)$ are fresh.

Definition 5.3.2. Given an atom $g_1 = r(X_1, \ldots, X_n)$ and an atom $g_2 = r(Y_1, \ldots, Y_n)$, we say that g_1 and g_2 unify if there exists a variable substitution θ such that $\theta(g_1) = \theta(g_2)$. Each such a θ is called unifier. Moreover, if g_1 and g_2 unify, we denote as $mgu(g_1, g_2)$ a most general unifier of g_1 and g_2.

We are now ready to define the algorithm `Rewrite`. At first, SQL query is translated into conjunctive query using standard SQL-to-CQ algorithm. Then the `Rewrite` algorithm is applied. Note, that the order of the variables, which is the one given by the translation from SQL to CQ, must be considered.

In the algorithm, $q[g/g']$ denotes the query obtained from q by replacing the atom g with a new atom g'.

Informally, the algorithm `Rewrite` first reformulates the atoms of each query $q \in P'$ and produces a new query for each atom reformulation (step (a)). More precisely, if there exists an inclusion assertion I and a conjunctive query $q \in P'$ containing an atom g, then the algorithm adds to P' the query obtained from q by replacing g with $gr(g, I)$. For the step (b), the algorithm `Rewrite` for each pair of atoms g_1, g_2, that unify, computes the query $q' = \texttt{Reduce}(q, g_1, g_2)$, obtained from q by the following algorithm:

Informally, the algorithm `Reduce` starts by eliminating g_2 from the query body. Then the substitution $mgu(g_1, g_2)$ is applied to the whole query (both the head and the

Algorithm 4: `Rewrite`$(q, \mathcal{S})$: Rewrites a conjunctive query into a union of conjunctive queries

Input: Conjunctive query q, system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$
Output: Union of conjunctive queries P

```
P := {q};
repeat
    P' := P;
    foreach q ∈ P' do
        (a) foreach g in q do
            foreach I in K do
                if I is applicable to g then
                    P := P ∪ q[g/gr(g, I)];
                end
            end
        end
        (b) foreach g1, g2 in q do
            if g1 and g2 unify then
                P := P ∪ {reduce(q, g1, g2)};
            end
        end
    end
until P' = P ;
return P;
```

Algorithm 5: `Reduce`(q, g_1, g_2): Reduces a conjunctive query by eliminating unifying atoms.

Input: Conjunctive query q, atoms $g_1, g_2 \in body(q)$
Output: Reduced conjunctive query q'

```
q' := q;
σ := mgu(g1, g2);
body(q') := body(q') - {g2};
q' := σ(q');
return q';
```

body). Due to the unification, variables that were bound in q may become unbound in q'. Hence, the inclusion assertions that were not applicable to atoms of q may become applicable to atoms of q' (in the next executions of step (a) of algorithm `Rewrite`). Function σ applied to q' replaces with _ each unbound variable in q'.

In order to compute the answers of q to $\mathcal{S}$, we need to evaluate the set of conjunctive queries P produced by the algorithm `Rewrite`. Every query q in P is transformed into an SQL query. The algorithm `Answer`, given a satisfiable KB $\mathcal{K}$ and a query q, computes the answer to q over $\mathcal{K}$. `Eval`$(q, \mathcal{D})$ denotes the evaluation of the SQL query q over the database $\mathcal{D}$.

Algorithm 6: `Answer`$(q, \mathcal{S}, \mathcal{D})$: Computes the answers of a conceptual query q over the database $\mathcal{D}$.

Input: Conceptual query q, system $\mathcal{S} = \langle \mathcal{K}, \mathcal{R}, \mathcal{M} \rangle$, database $\mathcal{D}$ for relational schema $\mathcal{R}$
Output: $ans(q, \mathcal{S}, \mathcal{D})$

$\mathcal{K} :=$ `Normalize`$(\mathcal{K})$;
return `Eval`(SQL(`Rewrite`$(CQ(q, \mathcal{S}), \mathcal{S})$,$\mathcal{R}$),$\mathcal{D}$);

5.4 Query Answering: Case Study

The approach of query answering by rewriting described in this chapter can be immediately applied to the setting analyzed and defined in previous chapters. That is, the relations described in a system $\mathcal{S}$ can be described by means of views instead of being actual tables. Thus, after extracting the set of views from a database (chapter 4), we have the setting where the underlying DBMS contains a view for each concept in a *DLR-Lite* KB. The case study of all the process is analyzed in the following.

5.4.1 Formalizing the System

We consider the part of the example database defined in the case study of section 3.2 of chapter 3. Since we have already shown the process of view extraction in example 4.3.1 (which actually uses the same example database), we skip this part here. In order to exhibit the query answering process and to provide a clear intuition, we display graphically the extracted views by means of tables. Additionally, suppose the answers to the views (i.e., data instances, called extensions of the views) are as displayed in figure 5.1[3].

The extraction methodology defined in section 4.3, together with the set of views identifies the elements of ER schema, decides on cardinality constraints for entities participating in a relationship, as well as finds the constraints for IS-A relationships. As already discussed before, we model all the identified elements and the constraints with *DLR-Lite*. For the sake of clarity, the names of the constructs in ER diagram and

[3]We assume without loss of generality that data instances already appear in the database.

$\mathsf{Employee}_\mathsf{V}$

SSN	EName	BDate
'1122'	'peter'	01/01/1980
'3344'	'susan'	02/05/1970
'5566'	'anna'	12/06/1982
'7788'	'bob'	28/02/1975

$\mathsf{AssignedTo}_\mathsf{V}$

SSN	Code	Hours
'1122'	'AN05'	160
'3344'	'AN05'	250
'5566'	'AN05'	80
'7788'	'CB06'	300

$\mathsf{Project}_\mathsf{V}$

Code	Title
'AN05'	'anaconda'
'CB06'	'cobis'

$\mathsf{Institute}_\mathsf{V}$

InstId	Name
'001'	'IM'
'002'	'SD'
'003'	'DB'

$\mathsf{DevelopedWith}_\mathsf{V}$

Code	InstId
'AN05'	'001'
'CB06'	'002'

$\mathsf{Permanent}_\mathsf{V}$

SSN	MonthlySalary
'1122'	2000
'3344'	2500
'7788'	1800

$\mathsf{Consultant}_\mathsf{V}$

SSN	DailyWage
'5566'	95

Figure 5.1: Views of the example

concepts in *DLR-Lite* language will be the same. Thus, the entities and relationships identified in example 4.3.1 correspond to the following atomic concepts: Employee, Project, Institute, Permanent and Consultant, and the relationships AssignedTo between Employee and Project, and DevelopedWith between Project and Institute. Note that we fix the order of arguments.

The component position for the relationships in *DLR-Lite* correspond to the order of arguments in the relationship of the ER schema. For instance, for the binary relationship AssignedTo between Employee and Project, the projection over the first (resp. second) component of the relationship AssignedTo will be denoted as $\exists[1]$AssignedTo (resp. $\exists[2]$AssignedTo).

The expressivity of *DLR-Lite* lies in its ability to capture the constraints used in conceptual data modeling. We model these constraints in terms of *DLR-Lite* assertions. For ease of readability, we also display the ER diagram in figure 5.2, which models the ER elements and constraints extracted with a methodology of section 4.3. Thus, we list the assertions below:

(1) $\exists[1]\mathsf{AssignedTo} \sqsubseteq \mathsf{Employee}$

(2) $\exists[2]\mathsf{AssignedTo} \sqsubseteq \mathsf{Project}$

(3) $\exists[1]\mathsf{DevelopedWith} \sqsubseteq \mathsf{Project}$

(4) $\exists[2]\mathsf{DevelopedWith} \sqsubseteq \mathsf{Institute}$

(5) $\mathsf{Permanent} \sqsubseteq \mathsf{Employee}$

(6) $\mathsf{Consultant} \sqsubseteq \mathsf{Employee}$

(7) Permanent `disj` Consultant

(8) $\mathsf{Employee} \sqsubseteq \exists[1]\mathsf{AssignedTo}$

(9) $\mathsf{Project} \sqsubseteq \exists[2]\mathsf{AssignedTo}$

(10) $\mathsf{Project} \sqsubseteq \exists[1]\mathsf{DevelopedWith}$

(11) (`funct` $\exists[1]$AssignedTo)

(12) (`funct` $\exists[1]$DevelopedWith)

(13) (`funct` $\exists[2]$DevelopedWith)

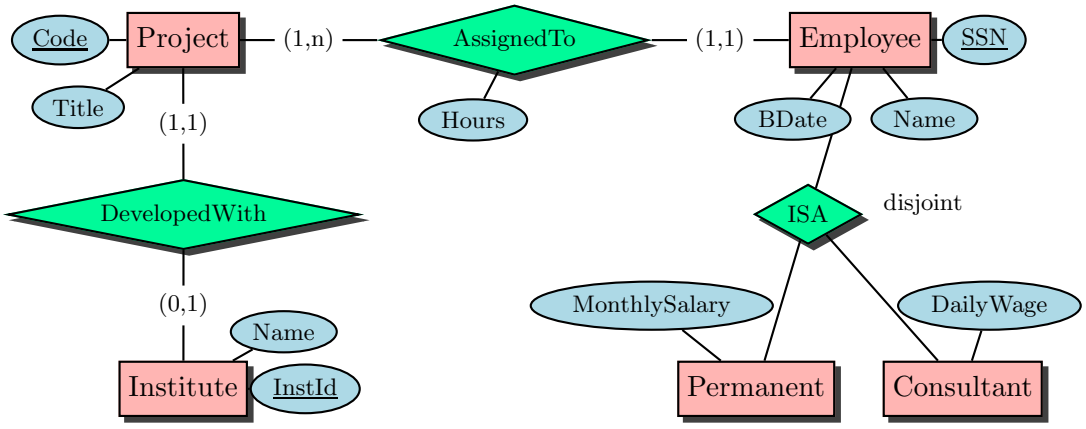


Figure 5.2: ER diagram of the example

In particular, we identified role-typing, participation constraints, IS-A, disjointness and functionality restriction constructs. Assertions (1) to (4) correspond to role-typing of relationships AssignedTo and DevelopedWith. Assertions (5) and (6) express IS-A relationship between Permanent and Employee, and Consultant and Employee, respectively, while (7) states Permanent and Consultant to be disjoint. Assertions (8) to (10) state mandatory participation for entities Employee and Project to the relationship AssignedTo, and for entity Project to the relationship DevelopedWith. Finally, functionality restrictions in (11) - (13) are used to impose the upper limit for the participation of Employee in the relationship AssignedTo (i.e., Employee can participate in AssignedTo at most once), and for the entities Project and Institute in the relationship DevelopedWith.

Having the setting defined above, we define the mapping between the conceptual level (*DLR-Lite* ontology) and the relational level (set of views). Since our views are defined in a way that every of them is in one-to-one correspondence with an element of an ontology, the mapping is straightforward. That is, to each concept in an ontology, mapping $\mathcal{M}$ associates a view over the sources. The next step consists in devising the appropriate components. We start from atomic concepts, for which we have a single component. Informally it means that the only component of the view corresponding to an atomic concept contains all attributes of that view. For n-ary relationships, we have n components, where each component corresponds to the role of the relationship. Consider, for instance, the view $\mathsf{AssignedTo}_\mathsf{V}$ which corresponds to the binary relationship AssignedTo between Employee and Project (the order of arguments is fixed). The view has 2 components, where the first contains the attributes providing information about employees (i.e. SSN) and the second contains the attributes providing information about projects (i.e. Code). The table 5.1 summarizes the associations between concepts and relationships in the conceptual level, and views over the relational level. The components are underlined.

Concept/ Relationship	**View**
Employee	$\mathsf{Employee_V}$(SSN, EName, BDate)
Project	$\mathsf{Project_V}$(Code, Title)
Institute	$\mathsf{Institute_V}$(InstId, IName)
Permanent	$\mathsf{Permanent_V}$(SSN, MonthlySalary)
Consultant	$\mathsf{Consultant_V}$(SSN, DailyWage)
AssignedTo	$\mathsf{AssignedTo_V}$(SSN, Code, Hours)
DevelopedWith	$\mathsf{DevelopedWith_V}$(Code, InstId)

Table 5.1: Mapping between conceptual and relational level

5.4.2 Query Answering

Once the mapping is defined, we are ready to formulate conceptual queries (over the ontology) and evaluate them using relational DBMS.

The first step that we have to perform according to the algorithm `Answer` is to normalize the knowledge base. It is easy to see that our KB does not contain such disjointness assertions that satisfy the inference rule for KB normalization. Next, suppose we want to know employees assigned to projects that are being developed in collaboration with an institute. We formulate the conceptual query as follows:

SELECT A.SSN
FROM AssignedTo **AS** A
JOIN DevelopedWith **AS** D
ON D.1 = A.2

In order to convert this conceptual query to a conjunctive query over the sources, we first translate it to a standard SQL SPJ query over the views. Thus, we get the following SQL query:

SELECT A.SSN
FROM $\mathit{AssignedTo_V}$ **AS** A
JOIN $\mathit{DevelopedWith_V}$ **AS** D
ON D.Code = A.Code

Next, since the algorithm makes use of conjunctive queries, the SQL SPJ query above is translated into an equivalent conjunctive query (we omit the details of translation, see [2]). Thus, we have the following conjunctive query:

$$q(x) \leftarrow \mathit{AssignedTo_V}(x, y, z), \mathit{DevelopedWith_V}(y, w)$$

Let us analyze the algorithm $\texttt{Rewrite}(q, \mathcal{S})$, where $q(x) \leftarrow \mathit{AssignedTo_V}(x, y, \_)$, $\mathit{DevelopedWith_V}(y, \_)$. At the first execution of step (a), the algorithm inserts in P the new query $q(x) \leftarrow \mathit{AssignedTo_V}(x, y, \_), \mathit{Project_V}(y)$, by applying to the atom $\mathit{DevelopedWith_V}(y, \_)$ the inclusion assertion (10). Then, at the second execution

of step (a), the query $q(x) \leftarrow AssignedTo_V(x, y, \_), AssignedTo_V(\_, y, \_)$ is added to P, according to application of the inclusion assertion (2) to the atom $Project_V(y)$. Since the two atoms of the second query unify, step (b) of the algorithm inserts the query $q(x) \leftarrow AssignedTo_V(x, \_, \_)$ into P. At a next iteration, step (a) produces the query $q(x) \leftarrow Employee_V(x)$, by applying the inclusion assertion (8) above to $AssignedTo_V(x, \_, \_)$, and then, at a further execution of step (a), it generates the queries $q(x) \leftarrow Permanent_V(x)$ and $q(x) \leftarrow Consultant_V(x)$, by applying assertions (5) and (6), respectively, to the atom $Employee_V(x)$. The set constituted by the above six queries and the original query q is then returned by the algorithm. We list them all below:

(Q1) $q(x) \leftarrow AssignedTo_V(x, y, \_), DevelopedWith_V(y, \_)$,

(Q2) $q(x) \leftarrow AssignedTo_V(x, y, \_), Project_V(y)$,

(Q3) $q(x) \leftarrow AssignedTo_V(x, y, \_), AssignedTo_V(\_, y, \_)$,

(Q4) $q(x) \leftarrow AssignedTo_V(x, \_, \_)$,

(Q5) $q(x) \leftarrow Employee_V(x)$,

(Q6) $q(x) \leftarrow Permanent_V(x)$,

(Q7) $q(x) \leftarrow Consultant_V(x)$

Next, in order to evaluate the set of conjunctive queries over the database, we need to convert them back to SQL. Again, we skip the conversion details and refer to [2] for more information. The translated queries are listed below (the order of queries is preserved).

```
(Q1) SELECT A.SSN
     FROM AssignedTo_V AS A
     JOIN DevelopedWith_V AS D
     ON D.Code = A.Code

(Q2) SELECT A.SSN
     FROM AssignedTo_V AS A
     JOIN Project_V AS P
     ON P.Code = A.Code

(Q3) SELECT A.SSN
     FROM AssignedTo_V AS A
     JOIN AssignedTo_V AS B
     ON B.Code = A.Code

(Q4) SELECT A.SSN
     FROM AssignedTo_V AS A
```

(Q5) **SELECT** E.SSN
FROM $Employee_V$ **AS** E

(Q6) **SELECT** P.SSN
FROM $Permanent_V$ **AS** P

(Q7) **SELECT** C.SSN
FROM $Consultant_V$ **AS** C

Finally, the last step is to evaluate these SQL queries over the set of our materialized views. The answers returned for the seven queries are displayed below.

(Q1)	(Q2)	(Q3)	(Q4)	(Q5)	(Q6)	(Q7)
'1122'	'1122'	'1122'	'1122'	'1122'	'1122'	'5566'
'3344'	'3344'	'3344'	'3344'	'3344'	'3344'	
'5566'	'5566'	'5566'	'5566'	'5566'	'7788'	
'7788'	'7788'	'7788'	'7788'	'7788'		

Clearly, the final answer is the union of the answers returned by every query. Hence, the SSN of employees assigned to projects that are developed in collaboration with an institute are the following:

answer
'1122'
'3344'
'5566'
'7788'

Chapter 6

Conclusions and Future Work

In this thesis we studied the problem of extracting and materializing the conceptual schema from a relational database.

The key discovery part of our topic benefits from a large body of work on reverse engineering of relational databases: many approaches have been designed for extracting the conceptual schema from a database. On the other hand, the motivating idea of our work - to exhibit to users a conceptual view of the data, and allow them to pose queries over such a conceptual view - is a relatively new research area.

Our approach comprises of three parts. First, the principles of mapping the ER model to relational tables are revised. Second, the reverse process of extracting the set of views and the conceptual model itself is defined. Finally, once the conceptual schema is obtained from a database, we describe the mechanism to access information with the mediation of an ontology and, in particular, the way of formulating and answering the queries posed in terms of the conceptual schema.

For constructing the views together with an ontology, we relied on standard database design principles from ER diagrams, that systematically uncover the connections between the constructs of relational schemas and those of conceptual schemas. However, we have found these principles unstructured and incomplete, in the sense that the constraints associated with ER objects are not taken into acncount while designing relations in relational schema. For this reason we have proposed a methodology for translating ER schemas into relational schemas, by fully exploiting the knowledge of ER schema in order to capture all the information, including primary and foreign keys, uniqueness constraints, constraints on null values, occurrence of values in a relation w.r.t. the referenced relation, etc. Even though the underlying ideas are well-known, our methodology is the first concrete proposal for this problem.

Based on the analysis of how ER constructs can be mapped to relational constructs, at the next step, we have devised the reverse structure extraction by constructing the set of views, under the assumption that the input relational database was generated by our proposed ER-to-Relational methodology. Every view over the actual data is constructed in such a way that it corresponds to a unique ER construct. Additionally, we have also identified the ER constructs themselves and presented algorithms for deciding on cardinality and participation constraints for entities participating in relationships, and for finding disjoint and/or covering constraints for IS-A relationships

of ER schema. In this setting, the vocabulary over the sources is seen as a set of views over the vocabulary of the conceptual schema.

Finally, we have described a query answering system that enables to pose queries over the conceptual schema of a database, re-expressing a conceptual query in terms of views over the underlying database and evaluating the rewriting over it. For this, we have devised a flexible way of mapping the conceptual level to the underlying relational level, which provides users an SQL-like query language over the conceptual schema. We have shown with a case study that the formalism *DLR-Lite* is expressive enough to capture most important ER constructs and that the query rewriting mechanism provides a great flexibility w.r.t. the data sources, making it applicable also to information integration scenarios.

The work presented in this thesis can be extended in several ways. One could investigate the constructs and constraints proposed for various versions of the ER model that were not considered in this thesis, e.g. multivalued attributes, aggregation, etc. Thus, our proposed ER-to-Relational methodology could be expanded to cover more cases. A possible expansion would be to add the analysis of linguistic relationships between keys of relations. In this way, the classification of relations would be more precise. However, one of the most interesting improvements would be to formalize the mapping rules of ER model to relational model with formulas expressed in logic. Then the correctness of relational representations for ER schemas could be proved by showing soundness and completeness of the mapping procedure.

Appendix A

Data for the Case Study

In this appendix we display the ER diagram and the set of SQL tables, resulting from the ER-to-Relational mapping methodology.

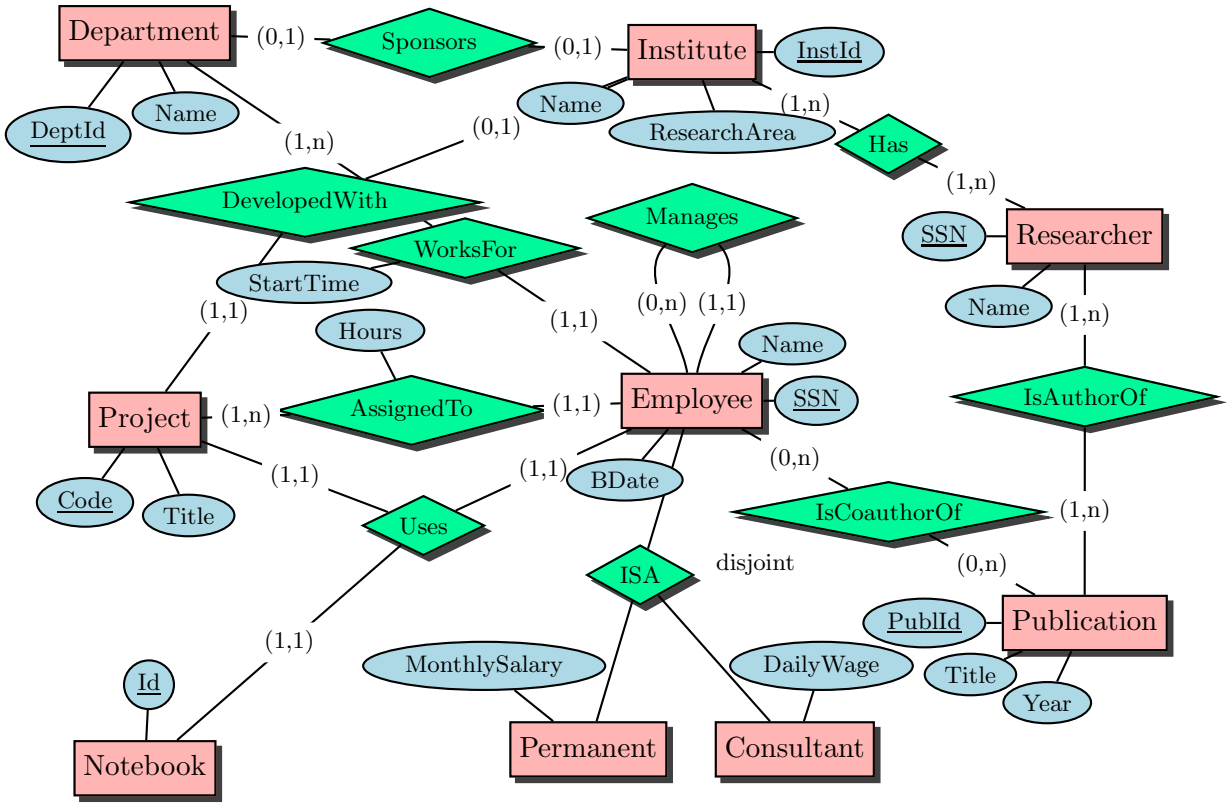


Figure A.1: Example ER diagram

```
CREATE TABLE Employee(
ESSN CHAR(10),
Name CHAR(50),
BDate DATE,
MngrSSN CHAR(10) NOT NULL,
DeptId INTEGER NOT NULL,
Code INTEGER NOT NULL,
StartTime DATE,
Hours INTEGER,
PTIMARY KEY (ESSN),
FOREIGN KEY (MngrSSN) REFERENCES Employee(ESSN),
FOREIGN KEY (DeptId) REFERENCES Department(DeptId),
FOREIGN KEY (Code) REFERENCES Project(Code));

CREATE TABLE Permanent(
ESSN CHAR(10),
MonthlySalary INTEGER,
PRIMARY KEY (ESSN),
FOREIGN KEY (ESSN) REFERENCES Employee(ESSN));

CREATE TABLE Consultant(
ESSN CHAR(10),
DailyWage INTEGER,
PRIMARY KEY (ESSN),
FOREIGN KEY (ESSN) REFERENCES Employee(ESSN));

CREATE TABLE Department(
DeptId CHAR(10),
Name CHAR(50),
InstId CHAR(10),
PRIMARY KEY (DeptId),
UNIQUE (InstId),
FOREIGN KEY (InstId) REFERENCES Institute(InstId));

CREATE TABLE Project(
Code CHAR(10),
Title CHAR(50),
InstId CHAR(10) NOT NULL,
PRIMARY KEY (Code));
UNIQUE (InstId),
FOREIGN KEY (InstId) REFERENCES Institute(InstId));

CREATE TABLE Notebook(
Id CHAR(10),
PRIMARY KEY (Id));

CREATE TABLE Institute(
InstId CHAR(10),
Name CHAR(50),
ResearchArea CHAR(10),
PRIMARY KEY (InstId));

CREATE TABLE Researcher(
RSSN CHAR(10),
Name CHAR(50),
PRIMARY KEY (RSSN));

CREATE TABLE Publication(
PublId CHAR(10),
Title CHAR(50),
Year INTEGER,
PRIMARY KEY (PublId));
```

```
CREATE TABLE Uses(
ESSN CHAR(10),
Code CHAR(10),
Id CHAR(10) NOT NULL,
PRIMARY KEY (ESSN, Code),
FOREIGN KEY (ESSN) REFERENCES Employee(ESSN),
FOREIGN KEY (Code) REFERENCES Project(Code),
FOREIGN KEY (Id) REFERENCES Notebook(Id));

CREATE TABLE Has(
InstId CHAR(10),
RSSN CHAR(10),
PRIMARY KEY (InstId, RSSN),
FOREIGN KEY (InstId) REFERENCES Institute(InstId),
FOREIGN KEY (RSSN) REFERENCES Researcher(RSSN));

CREATE TABLE IsAuthorOf(
RSSN CHAR(10),
PublId CHAR(10),
PRIMARY KEY (RSSN, PublId),
FOREIGN KEY (RSSN) REFERENCES Researcher(RSSN),
FOREIGN KEY (PublId) FOREIGN KEY Publication(PublId));

CREATE TABLE IsCoauthorOf(
ESSN CHAR(10),
PublId CHAR(10),
PRIMARY KEY (ESSN, PublId),
FOREIGN KEY (ESSN) REFERENCES Employee(ESSN),
FOREIGN KEY (PublId) REFERENCES Publication(PublId));

CREATE ASSERTION RestrWorksFor CHECK(
(SELECT D.DeptId FROM Department D)
EXCEPT
(SELECT E.DeptId FROM Employee E) = NULL);

CREATE ASSERTION RestrAssignedTo CHECK(
(SELECT Pr.Code FROM Project Pr)
EXCEPT
(SELECT E.Code FROM Employee E) = NULL);

CREATE ASSERTION RestrHas1 CHECK(
(SELECT I.InstId FROM Institute I)
EXCEPT
(SELECT H.InstId FROM Has H) = NULL);

CREATE ASSERTION RestrHas2 CHECK(
(SELECT R.RSSN FROM Researcher R)
EXCEPT
(SELECT H.RSSN FROM Has H) = NULL);

CREATE ASSERTION RestrIsAuthorOf1 CHECK(
(SELECT R.RSSN FROM Researcher R)
EXCEPT
(SELECT A.RSSN FROM IsAuthorOf A) = NULL);

CREATE ASSERTION RestrIsAuthorOf2 CHECK(
(SELECT Pb.PublId FROM Publication Pb)
EXCEPT
(SELECT DISTINCT A.PublId FROM IsAuthorOf A) = NULL);
```

```
CREATE ASSERTION RestrUses1 CHECK(
(SELECT E.ESSN FROM Employee E)
EXCEPT
(SELECT DISTINCT U.ESSN FROM Uses U) = NULL);

CREATE ASSERTION RestrUses2 CHECK(
(SELECT Pr.Code FROM Project P)
EXCEPT
(SELECT DISTINCT U.Code FROM Uses U) = NULL);

CREATE ASSERTION RestrUses3 CHECK(
(SELECT N.Id FROM Notebook N)
EXCEPT
(SELECT DISTINCT U.Id FROM Uses U) = NULL);

CREATE ASSERTION RestrISA CHECK(
(SELECT P.ESSN FROM Permanent P)
INTERSECT
(SELECT C.ESSN FROM Consultant C) = NULL)
```

Figure A.2: ER-to-Relational mapping example

Bibliography

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *PODS '98: Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 254–263. ACM Press, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] R. Alhajj. Extracting the extended entity-relationship model from a legacy relational database. *Information Systems*, 26(6):597–618, 2003.

[4] J. F. Allen and A. M. Frisch. What's in a semantic network? In *Proceedings of the 20th Annual Meeting on Association for Computational Linguistics*, pages 19–27. Association for Computational Linguistics, 1982.

[5] Y. An, A. Borgida, and J. Mylopoulos. Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In *Int. Conf. on Ontologies, Databases and Applications of Semantics (ODBASE)*, pages 1152–1169, 2005.

[6] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[7] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design. An Entity-Relationship Approach*. Benjamin/Cummings Publishing Company, Inc., 1992.

[8] R. J. Brachman. On the epistemological status of semantic networks. In N. V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 3–50. Academic Press, 1979.

[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Dl-lite: Tractable description logics for ontologies. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[10] D. Calvanese and M. Lenzerini. On the interaction between isa and cardinality constraints. In *Proceedings of the 10th IEEE Int. Conference on Data Engineering (ICDE'94)*, pages 204–213. IEEE Computer Society Press, 1994.

[11] P. P.-S. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[12] R. H. L. Chiang, T. M. Barron, and V. C. Storey. Reverse engineering of relational databases: extraction of an eer model from a relational database. *Data and Knowledge Engineering*, 12(2):107–142, 1994.

[13] E. F. Codd. Further normalization of the data base relational model. *IBM Research Report*, RJ909, 1971.

[14] T. M. Connolly and C. E. Begg. *Database Systems*. Addison-Wesley, fourth edition, 2005.

[15] S. S. Cosmadakis and P. C. Kanellakis. Functional and inclusion dependencies - a graph theoretical approach. In P. C. Kanellakis and F. P. Preparata, editors, *Advances in Computing Research, Vol. 3*, pages 163–184. JAI Press, 1986.

[16] C. J. Date. *An Introduction to Database Systems*. Addison-Wesley, eight edition, 2004.

[17] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, fourth edition, 2004.

[18] C. Fahrner and G. Vossen. A survey of database design transformations based on the entity-relationship model. *Data and Knowledge Engineering*, 15(3):213–250, 1995.

[19] S. Ferg. Cardinality concepts in entity-relationship modeling. In *Proceedings of the 10th Int. Conference on the Entity-Relationship Approach (ER'91)*, pages 1–30, 1991.

[20] R. Fikes and T. Kehler. The role of frame-based representation in reasoning. *ACM Communications*, 28(9):904–920, 1985.

[21] E. Franconi, S. Tessaris, B. C. Grau, B. Suntisrivaraporn, C. Lutz, R. Moller, and D. Lembo. Draft ontology task handbook. Deliverable D03, TONES EU-IST STREP FP6-7603, March 2006.

[22] P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity constraints: Semantics and applications. In *Logics for Databases and Information Systems*, pages 265–306, 1998.

[23] J. Grant and J. Minker. Numerical dependencies. In H. Gallaire, J. Minker, and J. M. Nicolas, editors, *Advances in Database Theory II*. Plenum Publ. Co, 1984.

[24] J. L. Hainaut. Database reverse engineering: Models, techniques and strategies. In *Proceedings of the 10th Conference on ER Approach*, 1998.

[25] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[26] J. Heflin and J. Hendler. A portrait of the semantic web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[27] J. Lee, K. Siau, and S. Hong. Enterprise integration with erp and eai. *Communications of the ACM*, 46(2):54–60, 2003.

[28] D. Lembo, M. Lenzerini, and R. Rosati. Methods and techniques for query rewriting. Deliverable D5.2, INFOMIX IST-2001-33570, April 2004.

[29] D. Lembo, C. Lutz, and B. Suntisrivaraporn. Tasks for ontology design and maintenance. Deliverable D05, TONES EU-IST STREP FP6-7603, May 2006.

[30] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS02)*, pages 233–246. ACM Press, 2002.

[31] M. Lenzerini and P. Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4):453–461, 1990.

[32] A. Y. Levy. Logic-based techniques in data integration. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 575–595. Kluwer Academic Publishers, 2000.

[33] V. M. Markowitz and J. A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering*, 16(8):777–790, 1990.

[34] V. M. Markowitz and A. Shoshani. Representing extended entity-relationship structures in relational databases: A modular approach. *ACM Transactions on Database Systems (TODS)*, 17(3):423–464, 1992.

[35] M. R. Quillian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 216–270. The MIT Press, 1968.

[36] C. S. Dos Santos, E. J. Neuhold, and A. L. Furtado. A data type approach to the entity-relationship model. In P. P. Chen, editor, *Entity-Relationship Approach to Systems Analysis and Design. Proc. of the First International Conference on the Entity-Relationship Approach*, pages 103–119. North-Holland, 1979.

[37] T. Teorey, S. Lightstone, and T. Nadeau. *Database Modeling and Design*. Morgan Kaufmann Publishers, fourth edition, 2006.

[38] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, 1986.

[39] L. Tucherman, M. A. Casanova, P. M. Gualandi, and A. P. Braga. A proposal for formalizing and extending the generalization and subset abstractions in the entity-relationship model. In F. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying. Proc. of the Eight International Conference on Enity-Relationship Approach*, pages 29–41. North-Holland, 1989.

[40] J. D. Ullman. Information integration using logical views. In *ICDT '97: Proceedings of the 6th International Conference on Database Theory*, pages 19–40. Springer-Verlag, 1997.

[41] H. Wache, T. Vogele, U.Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hubner. Ontology-based integration of information - a survey of existing approaches. In *Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.

[42] X. Ye, C. Parent, and S. Spaccapietra. Cardinality consistency of derived objects in dood systems. In P. Loucopoulos, editor, *Proceedings of the 13th Int. Conference on the Entity-Relationship Approach (ER'94)*, volume 881 of *Lecture Notes in Computer Science*, pages 278–295. Springer, 1994.