# Relational Databases in RDF: Keys and Foreign Keys

Georg Lausen

University of Freiburg, Institute for Computer Science
Georges-Köhler-Allee, 79110 Freiburg, Germany
lausen@informatik.uni-freiburg.de

**Abstract.** Today, most of the data on the web resides in relational databases. To make the data available for the semantic web mappings into RDF can be used. Such mappings should preserve the information about the structure of keys and foreign keys, because otherwise important semantic information is lost. In this paper, we discuss several possible ways to map relational databases into an RDF graph. We discuss the problem of how to represent the original key and foreign key constraints in the resulting RDF graph and demonstrate, that different kinds of mappings require different solutions. We finally propose to explicitly represent the structure of keys and foreign keys by means of the vocabulary of a new RDF namespace.

## 1 Introduction

Today, most of the data on the web resides in relational databases. Typically, even in the case when several databases represent information about the same application domain, they use different schemata. The idea of the semantic web is to support semantic interoperability between programs exchanging data. To achieve this goal W3C has standardized several languages to define ontologies [17], which can be used to define common vocabularies and structures for certain applications. Prominent examples are the *Resource Description Framework* RDF [21], which is considered as the basis for building the semantic web, and the *Web Ontology Language* OWL, whose mostly considered variant is based on a Description Logic [18,1]. Using RDF, any kind of information can be represented by a set of so called triples, where each triple states a subject-property-object relationship. As each such triple can be understood as a directed edge from the subject to the object, where the edge is labelled with the respective property, instead of a set of triples a corresponding RDF graph is considered. Thus, exporting data from relational databases to the semantic web using RDF basically means to map the relational data into an RDF graph.

A relational database may be physically exported as an RDF graph according to some mapping, respectively mapped into RDF in a similar fashion to a virtual relational view definition. Several approaches to define such mappings have already been described [2,5,4,12], however much less attention has been payed on the question how key and foreign key expressions stated in the relational database can be represented in RDF. In [12,8] a mapping is proposed which aims at the representation of keys and foreign keys as well, however the proposed mapping is specific to mapping an Entity-Relationship Model into RDF.

If key and foreign key information is lost when mapping a relational database into an RDF graph, the representational quality of the result has degraded substantially. This may become an important issue,

– if a user builds her own knowledge base by integrating several RDF graphs found on the internet,
– if an exported RDF graph is imported in a relational database at another place,
– if updates on a materialized RDF graph have to be performed such that key and foreign key properties have to be checked.

Finally, if an RDF graph is upgraded to OWL, keys and foreign keys could become interesting for reasoning, as well.

In this paper we will discuss various ways to map a relational database into RDF. In principle, many different methods may be applied. For each such mapping we will discuss how key and foreign key constraints can be expressed. For some mappings this will be rather straightforward, while other mappings may require additional constraints, or will require rather contrived solutions. Complications will arise in situations where a key is built out of several attributes and some of them are used for a foreign key as well. In addition, RDF is proposed as a simple data model which allows anyone to make statements about any resource. Thus a mapping should not only allow to represent the constraints, however has to give credit to the philosophy of RDF as well. Based on these observations we consider the mapping problem from relational databases in RDF still to be not sufficiently understood.

As key and foreign key constraints can not be asserted inside RDF, we will finally argue that RDF should be extended by a vocabulary which allows the declaration of keys and foreign keys. When keys and foreign keys are explicitly stated as part of an RDF graph, RDF processors are enabled to check the corresponding constraints and in this way are able to guarantee important quality criteria of RDF graphs.

The structure of the paper is as follows. In Section 2 we shall introduce the basic formalism and a running examples which we will use to demonstrate the various mappings. Mappings from relational databases into RDF are presented and discussed in Section 3. In Section 4 we present a vocabulary which allows to express key and foreign key constraints inside an RDF graph. Section 5 finally concludes the paper.

## 2 Preliminaries

We shall first introduce the terminology we use in the sequel. We start with relational databases (cf. [10], Section 3). A relational database schema **R** is a set of relation schemata identified by $R$, $\mathbf{R} = (R_1, ..., R_n)$. We use $Att(R)$ to denote the set of attributes of the relation symbol $R$. An instance $\mathbf{I}$ of $\mathbf{R}$ is a tuple $(I_1, ..., I_n)$, where for $1 \leq i \leq n$ $I_i$ is a finite instance of $R_i$, i.e. a finite subset of the $n$-ary cartesian product over an underlying domain. An element $\mu \in I$ is called tuple. Let $A \in Att(R)$, we use $\mu.A$ to denote the value of the attribute $A$ of the tuple $\mu$.

A *key* over $\mathbf{R}$ is an expression of the form $R[A_1, ..., A_k] \rightarrow R$, where $R \in \mathbf{R}$ and for $1 \leq i \leq k$ it holds that $A_i \in Att(R)$.[1] Let $\mathbf{I}$ be an instance of $\mathbf{R}$. $\mathbf{I}$ satisfies $R[A_1, ..., A_k] \rightarrow R$ if and only if $\forall \mu_1, \mu_2 \in I$ $(\bigwedge_{1 \leq i \leq k}(\mu_1.A_i = \mu_2.A_i) \rightarrow \bigwedge_{A \in Att(R)}(\mu_1.A = \mu_2.A))$.

A *foreign key* over $\mathbf{R}$ is an expression of the form $R[A_1, ..., A_k] \subseteq R'[A'_1, ..., A'_k]$, where $R, R' \in \mathbf{R}$, $\{A_1, ..., A_k\} \subseteq Att(R)$, $\{A'_1, ..., A'_k\} \subseteq Att(R')$, and $R'[A'_1, ..., A'_k] \rightarrow R'$. $R$ is called *child* and $R'$ *parent* of the foreign key. $\mathbf{I}$ satisfies $R[A_1, ..., A_k] \subseteq R'[A'_1, ..., A'_k]$ if and only if $\mathbf{I}$ satisfies $R'[A'_1, ..., A'_k] \rightarrow R'$ and $\forall \mu_1 \in I$ $\exists \mu_2 \in I'$ $(\bigwedge_{1 \leq i \leq k} \mu_1.A_i = \mu_2.A'_i)$.

Next we introduce the required RDF terminology adapting definitions in [20] for our purposes. We consider a *vocabulary* $\mathcal{V} = (N_C, N_P)$, where $N_C$ is a finite set of *classes* and $N_P$ is a finite set of *properties*. Given a vocabulary $\mathcal{V}$, an *interpretation* $\mathcal{I} = (\Delta_I, \Delta_D, .^{I_C}, .^{I_P})$ of $\mathcal{V}$ is given as follows:

- $\Delta_I$ is a nonempty set, called *object domain*,
- $\Delta_D$ is a nonempty set, called the *data domain*, which we assume to be disjoint from $\Delta_I$, $\Delta_I \cap \Delta_D = \emptyset$,
- $.^{I_C}$ is the *class interpretation function* assigning to each class $C \in N_C$ a finite subset $C^{I_C} \subseteq \Delta_I$,
- $.^{I_P}$ is the *property interpretation function* assigning to each property $Q \in N_P$ a finite subset $Q^{I_P} \subseteq \Delta_I \times (\Delta_I \cup \Delta_D)$.

Based on a given interpretation we can introduce a corresponding RDF graph. In the RDF document [21] among the nodes of an RDF graph it is distinguished between RDF URI references and literals. Literals in our framework are the elements of the data domain. By requiring that the object domain and the data domain are disjoint, we assume that literal values are not identified by URIs [22]. Therefore, in an RDF graph, literals with identical value will be represented by different nodes. This fact makes the following definitions a bit more complicated.

Let $\mathcal{I} = (\Delta_I, \Delta_D, .^{I_C}, .^{I_P})$ be an interpretation. The *RDF graph* $G^I = (N^I, E^I)$ of $\mathcal{I}$ then is a directed labelled graph, where

- for the set of nodes $N^I$ we have $N^I = N_C \cup \{a, b \mid (a, b) \in Q^{I_P}, Q \in N_P, b \in \Delta_I\} \cup \{a, b_{a,Q} \mid (a, b) \in Q^{I_P}, Q \in N_P, b \in \Delta_D\}$, and
- for the set of labelled arcs $E^I$ we have $E^I = \{(a, Q, b) \mid (a, b) \in Q^{I_P}, b \in \Delta_I\} \cup \{(a, Q, b_{a,Q}) \mid (a, b) \in Q^{I_P}, b \in \Delta_D\} \cup \{(a, \texttt{rdf:type}, C) \mid a \in C^{I_C}, C \in N_C\}$.

[1] We consider for each schema only one key, say the primary key.

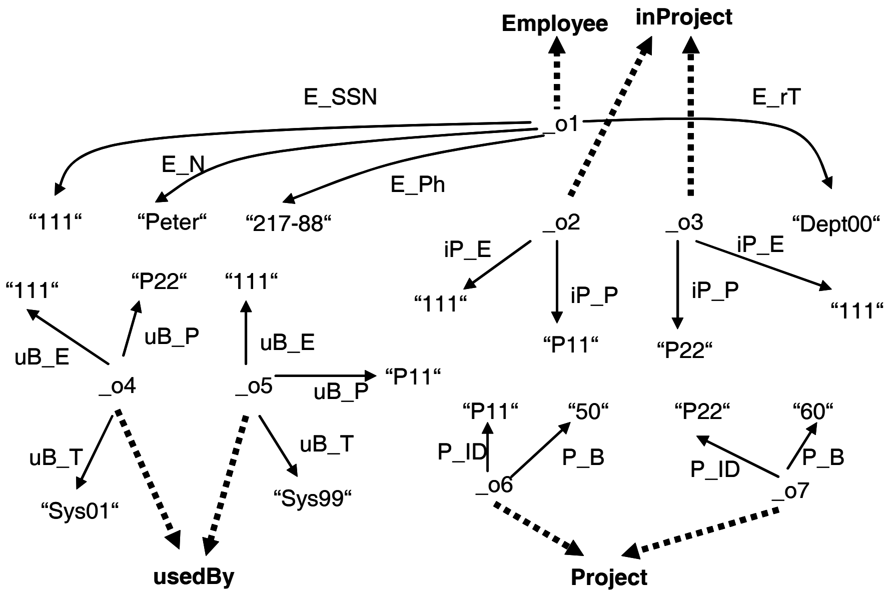

**Fig. 1.** Tuple-based mapping. Classes are indicated by bold font and typing by dotted edges.

We exemplify our discussion by means of a small running example. Consider the following relational schemata:

Employee(**SSN**, Name, Phone, reportsTo)
Project(**ID**, Budget)
inProject(**Empl**, **Proj**)
usedBy(**Tool**, **Proj**, Empl)

To indicate the key of a schema we use bold font. The schemata represent a scenario in which employees are described by name, ID, phone, and by the department they report to. Employees may be involved in projects and employees may use tools, where for a certain tool with respect to a certain project there is at most one employee who is using it.

We observe that `Empl` and `Proj` are foreign keys in schema `inProject`. Moreover, `Empl` and `Proj` together form a foreign key in schema `usedBy` with respect to `inProject`. For concreteness, we consider the following instances:

Employee

| **SSN** | Name | Phone | reportsTo |
|---|---|---|---|
| 111 | Peter | 217-88 | Dept00 |

Project

| **ID** | Budget |
|---|---|
| P11 | 50 |
| P22 | 60 |

inProject

| **Empl** | **Proj** |
|---|---|
| 111 | P11 |
| 111 | P22 |

usedBy

| **Tool** | **Proj** | Empl |
|---|---|---|
| Sys99 | P11 | 111 |
| Sys01 | P22 | 111 |

A simple kind of mapping from a relational database into RDF is as follows. Let $R$ be a relation schema, where $Att(R) = \{A_1, \ldots, A_k\}$. Introduce a class $C_R$ and properties $Q_{R,A_1}, \ldots, Q_{R,A_k}$. Let $I$ be an instance of $R$. For every tuple $\mu = (a_1, \ldots, a_k)$ in $I$ introduce a unique blank node $\_n_\mu$ and a labelled edge $(\_n_\mu, \texttt{rdf:type}, C_R)$. The naming of such blank nodes is arbitrary; for simplicity we can think of a numbering scheme based on incrementing a counter to assign to every blank node a unique number. For every nonnull value $\mu.A$ of $\mu$, $A \in Att(R)$, introduce an edge $(\_n_\mu, Q_{R,A}, (\mu.A)_{\_n_\mu, Q_{R,A}})$.[2] This mapping treats all tuples in the relation instances separately; we shall call the mapping *tuple-based*. Applying the tuple-based mapping we will get the RDF graph depicted in Figure 1 in which all property values are taken from the data domain, i.e. represented by literals in quotation marks. This mapping totally ignores that foreign key values also appear as key values and therefore refer to the same object. A tuple-based mapping thus has the following two severe deficiencies. It produces a lot of redundancies as foreign key values appear separate to key values and there is nothing in the graph which explicitly relates objects according to a foreign key relationship. Therefore, tuple-based mappings will not be considered further. The mappings we shall introduce in the next section demonstrate that these deficiencies result from an imperfect modelling and that they are not inherent to RDF.

## 3 Mapping Relational Databases into RDF

There have been several approaches described in the literature, which elaborate on the mapping of relational databases to an RDF graph (e.g. [2,5,4,12]). In this section we will discuss mappings with an eye on constraints. We will analyze possible ways to express the relational key and foreign key constraints in the resulting RDF graph. The following definitions clarify our notions of relational key and foreign key in the context of RDF.

To define a key constraint for RDF we first require that each property is interpreted by a (total) function. Thus, whenever a property $R \in N_P$ is involved in a key constraint, we require an additional functionality constraint to be given. Let $Q \in N_P$. We write $Func(C, Q)$ to state that for any object $o \in C$, on which a property $Q$ is defined, $Q$ associates $o$ with exactly one other object. We write $Key(C, Q_1, \ldots, Q_n)$ to state a relational key of class $C$ over properties $Q_1, \ldots, Q_n$ and we write $FK(C, [Q_1, \ldots, Q_n], C', [Q'_1, \ldots, Q'_n])$ to state a relational foreign key over the respective properties of child $C$ and parent $C'$.

**Definition 1.** *Let $\mathcal{I} = (\Delta_I, \Delta_D, .^{I_C}, .^{I_{Po}})$ be an interpretation. Let $C \in N_C$, $Q, Q_i, Q'_i \in N_P$, $1 \leq i \leq n$. Let $\psi$ be a constraint. $\mathcal{I}$ satisfies $\psi$, $\mathcal{I} \models \psi$, if there holds:*

- *Let $\psi$ be a* functionality constraint $Func(C, Q)$.
  $\{x \mid \#\{y \mid (x, y) \in Q^{I_P}\} \neq 1, x \in C^{I_C}\} = \emptyset$.

[2] A discussion how to represent a null value in RDF is beyond the scope of this paper.

– *Let* $\psi$ *be a* relational key constraint $Key(C, Q_1, \ldots, Q_n)$ *and let* $\mathcal{I} \models Func(C, Q_i)$, $1 \leq i \leq n$.
  If $\exists o_1, o_2 \in C^{I_C}$ such that $\exists v_i \in \Delta_I \cup \Delta_D, 1 \leq i \leq n$,
  where $(o_1, v_i), (o_2, v_i) \in Q_i^{I_P}$, then $o_1 = o_2$.
– *Let* $\psi$ *be a* relational foreign key constraint $FK(C, [Q_1, \ldots, Q_n], C', [Q'_1, \ldots, Q'_n])$ *and let* $\mathcal{I} \models Key(C', Q'_1, \ldots, Q'_n)$, $1 \leq i \leq n$.
  If $o_1 \in C^{I_C}$ then $\exists o_2 \in C'^{I_C}$ such that
  $(o_1, v_i) \in Q_i^{I_P}$ implies $(o_2, v_i) \in Q_i'^{I_P}, 1 \leq i \leq n$.

Keys and foreign keys according to these definitions are called *relational* to distinguish them from more generally defined keys. It is required that objects can be uniquely identified by properties directly related to them. Keys in a more general sense [14] could also be defined by means of path expressions in which the uniquely identifying property is defined by the composition of functional properties. We will return to this issue later again.

## 3.1 Value-Based Mappings

There are several ways to improve the tuple-based mapping. As a first approach, we represent key attribute values by URIs[3] and now are able to reference the

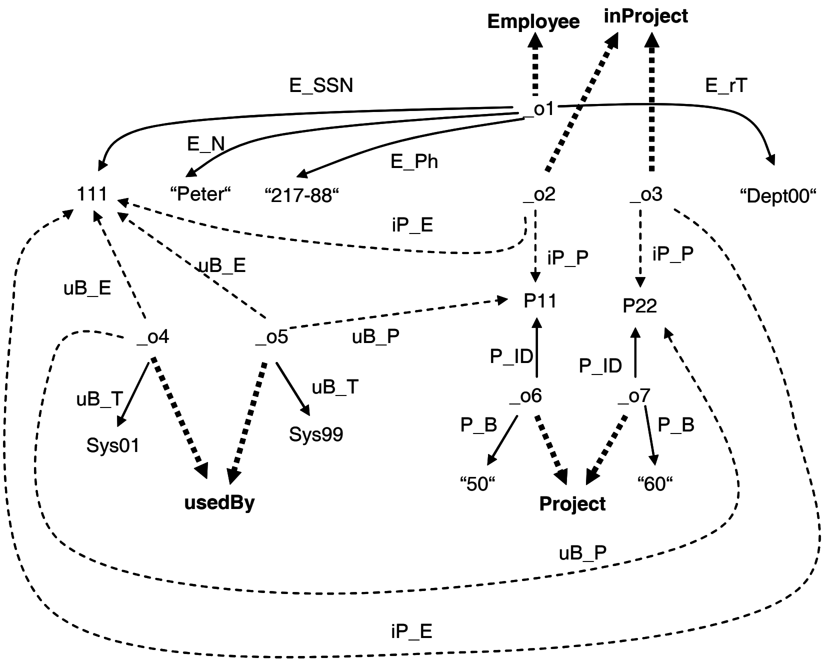

**Fig. 2.** Value-based mapping. Values of key attributes are URIs. References to foreign key attribute values are represented by dashed edges.

[3] For space reasons and not to overload our notation we consider only relative URIs in this paper.

key attribute values of parent objects from the respective child objects. This is demonstrated by Figure 2. Mappings following this approach are called *value-based* mappings. For example, properties `IP_P` and `iP_E` of the objects in class `inProject` now directly refer to the key values of the respective objects of class `Employee` and `Project`. Analogously, with respect to the foreign keys of the objects in class `usedBy`, we now directly refer to the respective key values of the objects in class `Employee` and `Project`.

Value-based mappings preserve the key and foreign key constraints that have been defined in the source relational database, because every key and foreign key stated in the relational database can be expressed by a relational key and foreign key constraint over the resulting RDF graph. We will next show, that such a direct correspondence does not immediately hold for other mappings, whose application seems to be quite natural in the context of RDF.

## 3.2 URI-Based Mappings

A popular mapping approach is based on encoding key values in URIs such that each object can be uniquely identified (cf. [4,5]) and then be further described by its properties. RDF triples that express links between such URIs express nothing else than foreign key relationships. At a first glance such an approach seems to solve the representation problem of keys and foreign keys in RDF in a very elegant and concise way.

However, keys may be built out of several attributes which themselves may be even foreign key attributes, in general. This typically will happen when relations are used to represent relationships between objects. In the Employee-Project-inProject example, the key of relation `inProject` is given by the keys of relations `Employee` and `Project`; we thus assume that one employee may work for several projects and one project may be processed by several employees. Having formed the corresponding URIs out of key values, the original foreign key constraint now appears as a less explicit constraint on the URI string-values. For example, if employee `111` works for project `P11`, then this will give rise to a relationship between `111` and `P11`. Following the above sketched approach we would first introduce the two URIs `111, P11`, then a combined URI `111&P11` and then define triples `(111&P11, iP_E, 111)`, respectively `(111&P11, iP_P, P11)`. As we can see, the foreign key problem still exists; however the foreign key relationships cannot be expressed over triples, but over URI values. This means, whenever there exists a URI `111&P11` in the RDF graph, there must also exist a triple `(111&P11, iP_E, 111)` and we would not accept a triple `(111&P11, iP_E, 222)`. In the RDF standard [21] RDF is proposed as an open-world framework that allows anyone to make statements about any object (resource). If a user is going to make a statement about object `111&P11`, she would certainly expect that there exists an object `111` as well. This means that there are still constraints reflecting foreign key constraints which are to be guaranteed.

The technique we have just described is one of the mappings suggested in [4,5], where such URIs are formed using string concatenation. However the syntax of URIs gives us enough room to construct URIs from key values in a more

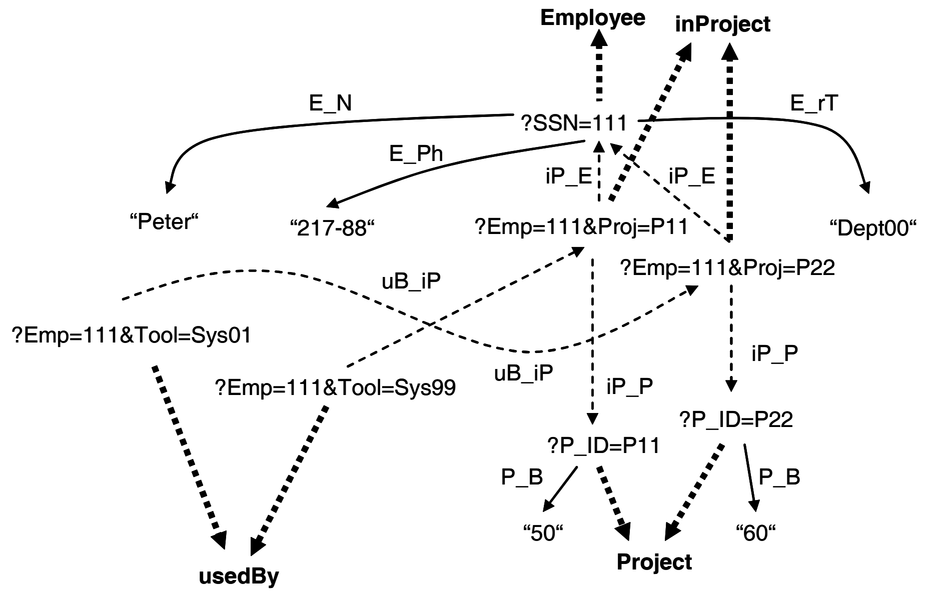

**Fig. 3.** URI-based mapping. Objects are represented by URIs which are build out of the corresponding key values. References between objects are represented by dashed edges.

standardized way. We propose to use the query part of an URI to build unique URIs by still making the origin of the components of such URIs visible. This is demonstrated in Figure 3. We call mappings following this idea *URI-based* mappings.

The discussion so far clearly shows that using key values to construct URIs representing an object may produce complications even though the approach looks attractive at first glance. When such URIs contain foreign key values, then a consistency problem arises which can only be solved by carefully checking the string representations of the URIs. For example, with respect to URI `?Emp=111&Tool=SYS99` representing an object of class `usedBy` and URI `?Emp=111&Proj=P11` representing an object of class `inProject` it has to be guaranteed that both `Emp`-parts are equal.

## 3.3 Object-Based Mappings

The next type of mappings identify objets by blank nodes and represent foreign key relationships by links between such blank nodes. We call this kind of mapping *object-based*; such mappings are also possible using the approach proposed in [4]. Different to the URI-based mappings, keys still are explicitly represented by properties. Therefore, in principle, key values can be represented by literals. However, as we will see later, there are reasons why it should be not only possible to reference objects, but key values as well. Therefore, key values are represented

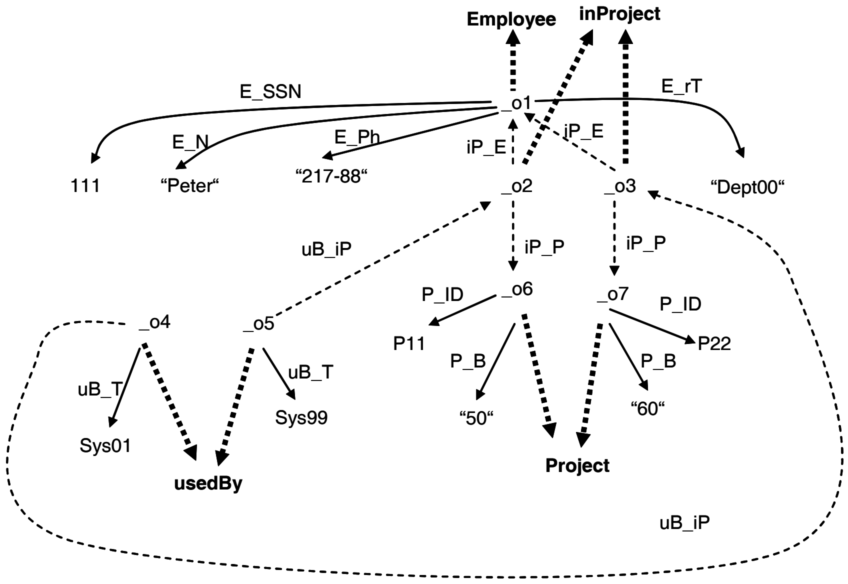

**Fig. 4.** Object-based mapping

by URIs in the sequel. As with the URI-based mappings there is no need to explicitly introduce properties for foreign keys - a link from the child object to its parent is sufficient. A mapping of this kind is shown in Figure 4.

As we can see, foreign key constraints are represented by links. However we need a constraint to ensures that such links will indeed exist. Such constraints are called *participation* constraints and are well-known from entity-relationship modelling (see also [15]). Let $C, C' \in N_C$ be classes and $Q \in N_P$ a property. Let $Q$ represent a foreign key relationship and consider $C$ as the child class and $C'$ the parent class. We write $Partcipate(C, Q, C')$ to state a participation constraint which can be formalized in our RDF setting as follows.

**Definition 2.** *Let* $\mathcal{I} = (\Delta_I, \Delta_D, .^{I_C}, .^{I_{Po}})$ *be an interpretation. Let* $\phi = Partcipate(C, Q, C')$*. We define* $\mathcal{I}$ *satisfies* $\phi$*,* $\mathcal{I} \models \phi$*, if there holds:*

$$\text{If } o_1 \in C^{I_C} \text{ then } \exists o_2 \in C'^{I_C} \text{ such that } (o_1, o_2) \in Q^{I_P}.$$

When carefully looking at Figure 4 the question arises where we have represented the key values of the objects of class `usedBy` with respect to the property giving us the ID of the project for which a certain employee uses a tool. As the key and the foreign key of class `usedBy` overlap, in some sense, the foreign key has stolen the key property giving us the project ID being part of the key. Therefore, the key for `usedBy` is not given by properties directly associated with `usedBy`, but given by property `uB_T` of `usedBy` and the path formed by the functional properties `uB_iP`, `iP_P` and `P_ID`. Thus, for being able to check key constraints, the notion of relational keys is too limited and a more general definition of key is in order. We write $Key(C, Q_1^\circ \dots Q_n^\circ)$ to state a general key constraint of class $C$ over property paths $Q_1^\circ, \dots, Q_n^\circ$, where each $Q_i^\circ$, $1 \leq i \leq n$ is a path of functional properties $Q_{ij} \in N_P$, $1 \leq i \leq n$, $1 \leq j \leq n_i$, $n_i \geq 1$, which is written $Q_{i1} \dots Q_{in_i}$.

**Definition 3.** *Let $\mathcal{I} = (\Delta_I, \Delta_D, .^{I_C}, .^{I_{Po}})$ be an interpretation. Let $C \in N_C$, $Q_i^\circ$ a path of functional properties $Q_{i1}, \dots, Q_{in_i}$, where $Q_{ij} \in N_P$, $1 \leq i \leq n$, $1 \leq j \leq n_i$, $n_i \geq 1$. Let $\psi$ be a* general key constraint $Key(C, Q_1^\circ, \dots, Q_n^\circ)$. *$\mathcal{I}$ satisfies $\psi$, $\mathcal{I} \models \psi$, if there holds:*[4]

$$\begin{array}{l}\text{If } \exists o_1, o_2 \in C^{I_C} \text{ such that } \exists v_i \in \Delta_I \cup \Delta_D, 1 \leq i \leq n, \\ \quad \text{where } (o_1, v_i), (o_2, v_i) \in Q_{i1}^{I_P} \circ \dots \circ Q_{in_i}^{I_P}, \text{ then } o_1 = o_2.\end{array}$$

General keys have been studied in description logics [14] and object-oriented databases [6,3]; the key concept as proposed by XML Schema [24] resembles general keys, as well. If we understand RDF as a formalism which has been introduced to allow to extend a given RDF graph by new information in a very flexible and open way, it is not clear to us, whether it is feasible to allow to define key properties by paths of properties. Keys seem to us much easier to comprehend when they are directly related with the respective objects. In Figure 5, to this end we introduce a property `uB_P` which applied on objects of class `usedBy` returns the missing `P_ID`-value such that the key `Tool, Proj` for objects of class `usedBy` now can be expressed by a relational key again. The price we have to pay is an additional constraint which guarantees that property `uB_P` and property path `uB_iP.iP_P.P_ID` will give the same value when applied on the same object. Such constraints have been called *subobjectproperty-chain* [19]. We write $SubPChain(C, Q_1, \dots, Q_n, S)$ to state a subobjectproperty-chain and formalize it according to our needs as follows:

**Definition 4.** *Let $\mathcal{I} = (\Delta_I, \Delta_D, .^{I_C}, .^{I_{Po}})$ be an interpretation. Let $\phi = SubPChain(C, R_1, \dots, R_n, S)$. We define $\mathcal{I}$ satisfies $\phi$, $\mathcal{I} \models \phi$, if there holds:*[5]

$$\begin{array}{c}\{(x, y) \mid (x, y) \in R_1^{I_P} \circ \dots \circ R_n^{I_P}, x \in C^{I_C}\} = \\ \{(x, y) \mid (x, y) \in S^{I_P}, x \in C^{I_C}\}.\end{array}$$

## 3.4 Discussion

In the preceding sections we have discussed various kinds of mappings from relational databases to RDF. This discussion does not claim to have been exhaustive. For example, [12] proposes a mapping in which attributes $A$ are represented by objects $o_A$ and each tuple $t$ of a (entity-) relation identifies a property $Q_t$ which then is used to assign $o_A$ some value $v$ out of the object or value domain. This would give rise to the triple $(o_A, Q_t, v)$. This technique is interesting as well and is based on the original formal definitions of the Entity-Relationship Model [9].

[4] The operator $\circ$ denotes the composition of binary relations.

[5] In contrast to [19] we can require equality of sets instead of a subset-relationship, because all properties involved are functional.

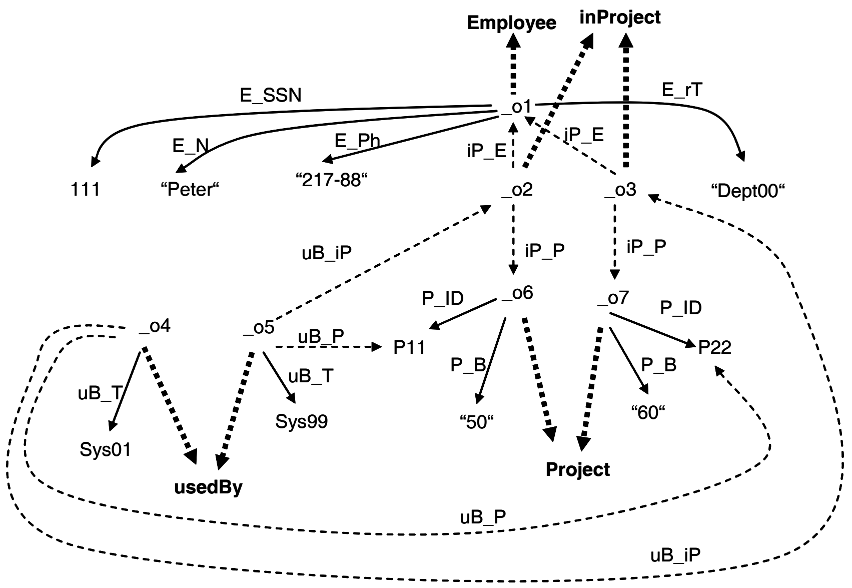

**Fig. 5.** Object-based mapping. To directly relate each object with its key properties, additional properties are introduced.

However, values may also appear as subjects in other triples whose predicates now do not correspond to tuples in a (entity-) relation, however to attributes of the given Entity-Relationship schema. Therefore, to our understanding, a resulting RDF graph is likely to be difficult to comprehend.

When mapping relational databases into RDF, we can choose mappings which preserve relational constraints and still use the linking capabilities of RDF (value-based mapping). However, we can also choose mappings which try to take the most advantage of the flexibility offered by RDF. Such mappings are attractive, but make it rather contrived to express key and foreign key constraints (URI-based mappings), or enforce the introduction of other kinds of constraints, which are not familiar in relational databases (object-based mapping). Which approach is the most appropriate one depends on the kind of applications for which a resulting RDF graph is assumed to be used. In the next section we will outline a technique that will allow us to state relational key and foreign key constraints as part of an RDF graph. A value-based mapping in conjunction with an explicit statement of the existing key and foreign key constraints inside the same RDF graph seems to us an attractive compromise between relational modelling and RDF modelling.

## 4 Keys and Foreign Keys Vocabulary for RDF

To get a clean and general solution for expressing keys and foreign keys inside an RDF graph we now propose to make that information explicit by means of an appropriate namespace. As a first idea one could use the XML-schema namespace [24], which allows to express keys and foreign keys, where the latter are called keyref. However, XML in contrast to RDF is a hierarchical model equipped with an implicit notion of order. Therefore, semantics of keys and foreign keys in XML and RDF are different, though similar in spirit. For these reasons we decided to extend the RDF vocabulary by a new dedicated namespace, which will be identified by prefix `rdfc`. This namespace extends the RDF vocabulary by two (meta-) classes `rdfc:Key` and `rdfc:FKey`, whose instances represent the key and foreign key definitions of the (application) classes. In addition, the namespace contains properties `rdfc:Key`, `rdfc:FKey` and `rdfc:Ref` which will allow to attach keys and foreign keys to classes and to associate foreign keys with the respective key of the parent class. The construction we shall outline is in the spirit of the `PRIMARY KEY` and `REFERENCE`-clause in SQL.

Keys and foreign keys may be built out of several components and therefore will be of type `rdf:Bag`. The construction is demonstrated in Figure 6. Starting from the graph depicted in Figure 2 we proceed as follows. We introduce objects `E_Key`, `P_Key`, `iP_Key` and `uB_Key`, respectively. In addition we introduce objects `iP_FKey1`, `iP_FKey2` and `uB_FKey` to represent the two foreign keys of class `inProject` and the foreign key of class `usedBy`. We add corresponding typing edges from the newly introduced objects to classes `rdfc:Key` and class `rdfc:FKey`, respectively. Next, we add the corresponding edges labelled with

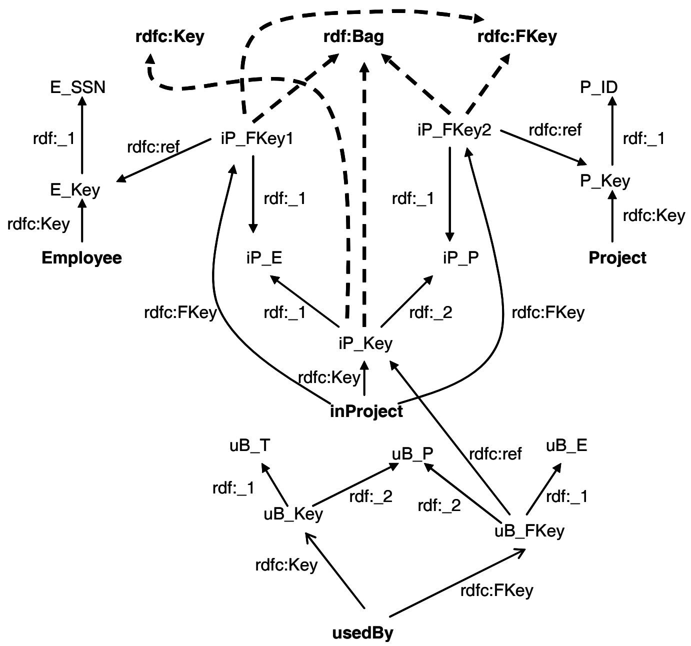

**Fig. 6.** Keys and foreign keys definitions are explicitly represented as part of an RDF graph. The typing of keys and foreign keys is visualized only for class `inProject`.

`rdfc:Key`, respectively `rdfc:FKey`, in order to relate a key and foreign key to its respective class, i.e. one class out of `Employee, Project, inProject, usedBy` for each key, respectively foreign key. In addition, for each foreign key we introduce an edge labelled `rdfc:ref` to associate the foreign key with the respective key of the parent class. Finally, as keys and foreign keys are of type `rdf:Bag` as well, we add corresponding typing edges and also indicate the components of keys and foreign keys by the edges labelled with `rdf:_1` and `rdf:_2`.

If we assume an RDF query language be given, which allows to traverse a RDF graph in edge and inverse edge direction, e.g. SPARQL [23], it is easy to see, how for each class its key and its foreign keys can be determined. Therefore, whenever we are interested in this information, e.g. for checking the constraints, we can first extract the constraints from the graph and then perform the checking. In [13] we show that all these steps can be performed using SPARQL.

## 5 Conclusion

In this paper we have argued to introduce key and foreign key constraints into RDF to explicitly state semantic information about the information represented by an RDF graph. Checking of the constraints is possible by testing appropriate SPARQL-queries for emptiness (cf. [13]), in a similar way to constraint checking using SQL. In [15,16,7] constraints are investigated with respect to OWL. In [15] the fundamental difference between constraints in relational databases and ontology languages is discussed and [16,7] present possible ways of integration. While these approaches emphasize the description logic point of view on constraints, we are interested in the question, how key and foreign key constraints can be expressed depending on the particular mapping applied from a relational database into an RDF graph. In addition, we show how these constraints can be expressed as part of an RDF graph. As a consequence, processing of the RDF data originating from relational data and RDF data originating from relational schema (meta-) data, in particular keys and foreign keys, becomes possible, e.g. using SPARQL or rule languages like F-Logic [11]. In contrast, the description logic based approaches allow query answering and powerful logical reasoning. Our approach to extend RDF by constraints and the description logic based approach can therefore be considered orthogonal and we expect that both will be useful for semantic web applications.

**Acknowledgement.** I wish to thank Wolfgang May and Michael Schmidt for their insightful comments on this paper.

## References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
2. Berners-Lee, T.: Relational Databases on the Semantic Web (1998), `http://www.w3.org/DesignIssues/RDB-RDF.html`

3. Biskup, J., Polle, T.: Adding inclusion dependencies to an object-oriented data model with uniqueness constraints. Acta Inf. 39(6-7) (2003)
4. Bizer, C.: D2R MAP - A Database to RDF Mapping Language. WWW (Posters) (2003)
5. Blakeley, C.: Mapping Relational Data to RDF with Virtuoso's RDF Views. OpenLink Software (2007)
6. van Bommel, M.F., Weddell, G.E.: Reasoning about equations and functional dependencies on complex objects. IEEE ToKDE 6(3) (1994)
7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Can OWL model football leagues?. In: OWLED 2007 (2007)
8. Chatterjee, N., Krishna, M.: Semantic Integration of Heterogeneous Databases on the Web. In: ICCTA 2007 (2007)
9. Chen, P.P.: The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. on Database Systems 1(1) (1976)
10. Fan, W., Libkin, L.: On XML integrity constraints in the presence of DTDs. J. ACM 49(3) (2002)
11. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages. J. ACM 42(4) (1995)
12. Krishna, M.: Retaining Semantics in Relational Databases by Mapping them to RDF. In: IAT Workshops 2006 (2006)
13. Lausen, G., Meier, M., Schmidt, M.: SPARQling Constraints for RDF. In: EDBT 2008 (to appear, 2008)
14. Lutz, C., Areces, C., Horrocks, I., Sattler, U.: Keys, Nominals, and Concrete Domains. J. Artif. Intell. Res. (JAIR) 23 (2005)
15. Motik, B., Horrocks, I., Sattler, U.: Bridging the Gap Between OWL and Relational Databases. In: WWW 2007 (2007)
16. Motik, B., Horrocks, I., Sattler, U.: Adding Integrity Constraints to OWL. In: OWLED 2007 (2007)
17. Staab, S., Studer, R. (eds.): Handbook on Ontologies. Springer, Heidelberg (2004)
18. OWL Web Ontology Language Reference. W3C (2004)
19. OWL 1.1 Web Ontology Language Overview. Editor's Draft, April 6 (2007), `http://webont.org/owl/1.1/overview.html`
20. OWL 1.1 Web Ontology Language Model-Theoretic Semantics. Editor's Draft, April 6 (2007), `http://webont.org/owl/1.1/semantics.html`
21. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C (2004)
22. RDF Semantics. W3C (2004)
23. SPARQL Query Language for RDF. W3C (2007)
24. XML Schema Part 0: Primer Second Edition. W3C (2004)