Appeared in: In: S. M. Embury, N. J. Fiddian, A. W. Gray, A. C. Jones (eds.), Advances in Databases, 16th British National Conf. on Databases, BNCOD 16, Cardiff, Wales, July 1998, Lecture Notes in Computer Science 1405, Springer-Verlag, Berlin, 1998.

Considering Integrity Constraints During Federated Database Design *

Stefan Conrad Ingo Schmitt Can Türker

Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
{conrad|schmitt|tuerker}@iti.cs.uni-magdeburg.de

Abstract. Correct transformations and integrations of schemata within the process of federated database design have to encompass existing local integrity constraints. Most of the proposed methods for schema transformation and integration do not sufficiently consider explicit integrity constraints. In this paper we present an approach to deal with integrity constraints. Our approach bases on the idea to relate integrity constraints to extensions. A set of elementary operations for schema restructuring and integration is identified. For these operations we define major rules for dealing with integrity constraints. By means of a small example we then demonstrate the application of these rules.

1 Introduction

In many large organizations, different legacy data management systems are maintained and operated. These data management systems and the data managed by them have developed independently from each other. Among these systems there are database management systems but also merely file-based systems differing in several aspects such as data model, query language, and system architecture as well as the structure and semantics of the data managed. In this context the term 'heterogeneity of the data management systems' is commonly used. As a rule, applications for specific needs continue to be based on such systems. More recent applications often require access to the distributed databases but their implementation fails due to heterogeneity. *Federated database systems* [SL90] are designed to overcome this problem by integrating data of legacy database systems only virtually. That is, 'only' the local schemata are integrated to a federated schema. Global applications can retrieve and update federated data through the federated schema.

The main task during the design of a federated database is the transformation and integration of existing local schemata into a federated schema. The transformation [EJ95] and integration [LNE89,SPD92,RPG95,SS96a] of local schemata into a federated schema must fulfill the following requirements:

1. For each query against a local schema there exists a global query against the federated schema returning the same query result. This requirement avoids loss of information during the federation.

* This work was partly supported by the German Federal State Sachsen-Anhalt under grant number FKZ 1987A/0025 and 1987/2527R.

2. Each global update or insert operation can be propagated to the local DAMS correctly. That is, the local schemata must not be more restrictive than the federated schema. Therefore, schema transformation and integration have to encompass integrity constraints. Global integrity constraints can be used to optimize global queries and to validate global updates. In addition to the transformed local integrity constraints the federated schema contains 'integration constraints' [BC86] stemming from inter-schema correspondence assertions [SPD92]. These integration constraints force global consistency.

The second requirement demands to transform integrity constraints during the federation design process. In our approach, we consider integrity constraints referring to class extensions. A typical schema conflict occurs if extensions of two local classes can overlap. This overlapping itself is an integration constraint. By adding such integration constraints, local integrity constraints can come into conflict. That is, for objects simultaneously stored in both classes different conflicting integrity constraints can be defined. The problem in this case is to find a federated schema fulfilling the requirements mentioned above. Various publications tackle this conflict. [BC86,Con86], for instance, describe the integration of views and name local schemata (views) without conflicting integrity constraints conflictfree schemata. In general, testing for conflictfreeness is undecidable. In those approaches only conflictfree schemata (views) can be integrated correctly. [RPG95,EJ95] show how to integrate schemata into a federated schema considering integrity constraints. Conflicting integrity constraints are not integrated. [VA96] distinguish between subjective and objective constraints. Subjective constraints are only valid in the context of a local database and need not to be dealt with during schema integration. In this way only objective and non-conflicting integrity constraints are integrated.

In [AQFG95] the integration of conflicting integrity constraints bases on a conflict classification. In some cases, the more relaxed constraint is taken over to the integrated class. In other cases, the local constraints are reformulated in an implication format and attached to the integrated class.

In contrast to the mentioned approaches we integrate integrity constraints basing on set operations on class extensions. Extensions of local classes have to be decomposed into disjoint base extensions and later to be composed to the federated schema (cf. [SS96a,SC97,CHS+97]). Local integrity constraints are related to class extensions. This idea is already mentioned shortly in [BCW91]. By decomposing and composing class extensions in a well-defined way, we are able to derive integrity constraints for integrated schemata (as well as for arbitrary views, e.g. external schemata) in a clean and logical sound way. We tackle the problem of conflictfreeness of local schemata in a different way than most other approaches mentioned before. It is possible to discover conflicting integrity constraints during database schema integration. Such conflicts are often a hint that there are other conflicts behind that, e.g. a semantic conflict or a name conflict which has not been discovered up to that moment.

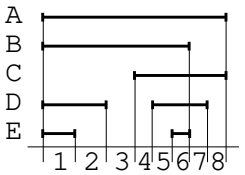
In this paper we focus on the integration of integrity constraints based on the integration of the extensions (classes) they belong to. Thereby, we continue the work presented in [CHS+97] where a basic analysis of the problem of integrating integrity constraints was given. The next section introduces elementary operations for decomposing and composing class extension. For these operations we present rules to transform and integrate integrity constraints. We briefly dis-

cuss the application of these rules to classes of frequently occurring integrity constraints before we give a detailed example.

The rules presented in this paper are not intended to be applied in a fully automatic manner. The designer of the federated database has to control the application of them. In this way he can see during the integration process whether there are conflicting constraints and in which way the integration has to be changed for resolving such conflicts.

2 Elementary Operations

During the schema integration process the main task is to find all relationships (integration constraints) between the class extensions in the base schemata which are to be integrated. Two class extensions can be always equal or always disjoint. In addition to these relationships, two class extensions can overlap semantically. A special case of overlapping class extensions is that one is always a subset of another one (cf. w.r.t. these relationships for instance [SPD92]). As showed in [SS96b], pairwise comparisons of class extensions is not always sufficient because not all pieces of information necessary for an adequate integration can be found in this way. Instead, all class extensions must be considered at the same time. The result of such an analysis can be presented in a graphical manner as depicted as follows:



The horizontal lines represent the class extensions of five classes A, B, C, D, and E, respectively. For instance, the class extensions of B and C are overlapping. Furthermore, E is a subclass of D. Based on this information, a set of *base extensions* can be identified as a canonical representation. In the figure, eight base extensions have been found and numbered by 1 to 8.

Using the standard set operations *intersection* and *difference* we can describe the base extensions in terms of the original class extensions. After such a decomposition into base extensions we can construct each possible class extension which might occur in the integrated schema by applying the standard *union* operation to a collection of base extensions. Based on this simple procedure of decomposing and then composing class extensions, resolving of extensional conflicts can be described in terms of the three standard set operations. In [SS96a] an algorithm is proposed to compose base extensions to a federated schema automatically. Due to the fact that *selection* is a very important operation for defining views or specifying export schemata, we also consider *selection* to be an elementary operation.

The decomposition of class extensions can produce base extensions which belong to local classes from different databases. Objects of such base extensions are managed by more than one database system simultaneously. The values of common attributes are stored redundantly. If the local databases store the current state of real-world objects correctly then the values of the common at-

tributes would be equal. Otherwise we can state that different values indicate inconsistency.

For two local classes of which the extensions overlap conflicting integrity constraints concerning common attributes can exist. For instance, the attribute salary can be restricted by 'salary $>$ 3000' in one database and by 'salary $>$ 2500' in another database. Such conflicting integrity constraints defined for common attributes of objects stored in both databases are indications of

1. wrong specification of extensional overlappings or
2. wrong specification of attribute-attribute relationships (attribute conflicts) or
3. incomplete or incorrect design of at least one local schema.

A wrong analysis of the schemata to be integrated (first and second item) can be corrected without violation of local autonomy. A wrong design of a local schema is more tricky. If both databases are consistent, i.e. the values of common attributes are equal, then the integrity constraints of both classes hold simultaneously. In this case the local integrity constraints can be corrected without invalidating the existing database. The corrected local integrity constraints would then help to detect wrong local updates.

However, sometimes the local database should continue in possibly inconsistent states due to the demand for design autonomy. A solution then is to regard common attributes as semantically different. In this way, the integrity constraints are not longer in conflict.

The next section presents rules for treating integrity constraints during the decomposition and composition of class extensions.

3 Rules for Dealing with Integrity Constraints

For presenting the rules which can be applied to integrity constraints during schema integration we first need to introduce our notation and to define some important terms.

As already mentioned before, integrity constraints are usually given for a certain set of objects. Therefore, we always consider constraints together with the sets of objects for which they are valid. We denote by $\Theta_\phi(E)$ that the constraint ϕ holds for a set E of objects. Typically, E is a class extension. From a logical point of view, $\Theta_\phi(E)$ says that all free variables occurring in ϕ are bound to the range E. The range E is also called the *validity range* of the constraint ϕ.

For simplifying the presentation we distinguish between two basic kinds of constraints. An *object constraint* is a constraint which can be checked independently for each object in the validity range of that constraint. Integrity checking for object constraints is easy and can be done very efficiently. In case we insert a new object or modify an existing object, we only need to consider that object for checking object constraints.

In contrast, a *class constraint* can be characterized as a constraint which can only be checked by considering at least two objects out of its validity range. Often, all currently existing objects in the validity range are needed for checking a class constraint.[1] Obviously, it is possible to distinguish several different kinds

[1] The distinction between object constraints and class constraints follows the taxonomy of integrity constraints presented in [AQFG95].

of class constraints, but, for the purposes of this paper, we do not need such a distinction.

In the following, we consider the four elementary operations identified in the previous section. For each elementary operation we give a rule for dealing with object constraints. Afterwards, we briefly discuss in which cases these rules can also be applied to class constraints.

Selection. Applying a selection σ_ψ (where ψ is the selection condition) to an extension E results in a subset of E denoted by $\sigma_\psi(E)$. The selection condition ψ can be considered as an object constraint. In consequence, ψ holds for the extension $\sigma_\psi(E)$. Assuming that an object constraint ϕ holds for E, we can conclude that $\phi \wedge \psi$ holds for $\sigma_\psi(E)$:

$$\Theta_\phi(E) \rightarrow \Theta_{\phi \wedge \psi}(\sigma_\psi(E)).$$

Difference. Applying the difference operation results in a subset of the first operand. Therefore, the difference operation and taking a subset can be dealt with in the same way with respect to object constraints. If an object constraint ϕ holds for an extension E, ϕ also holds for each subset of E. This is due to the fact that ϕ is an object constraint which can be checked independently for each object in E and this is true for each subset of E as well. This property leads to the following rule (where $E = E_1 \cup E_2$ and, thus, E_1 is a subset of E):

$$\Theta_\phi(E_1 \cup E_2) \rightarrow \Theta_\phi(E_1).$$

Intersection. The intersection of two (or more) extensions is always a subset of each of the original extensions. Due to the fact that object constraints can be transferred from an extension to any subset of that extension (as explained before), the following rule holds for object constraints. If an object constraint ϕ holds for an extension E_1 and an object constraint ψ holds for an extension E_2, then both ϕ and ψ (and thereby also the logical conjunction $\phi \wedge \psi$) hold for the intersection of E_1 and E_2:

$$\Theta_\phi(E_1), \Theta_\psi(E_2) \rightarrow \Theta_{\phi \wedge \psi}(E_1 \cap E_2).$$

Union. In case we unite two extensions E_1 and E_2 we can only use the knowledge that each object in $E_1 \cup E_2$ is an object in E_1 or in E_2. In consequence, if ϕ is an object constraint valid for E_1 and ψ an object constraint valid for E_2, then $\phi \vee \psi$ is valid for $E_1 \cup E_2$:

$$\Theta_\phi(E_1), \Theta_\psi(E_2) \rightarrow \Theta_{\phi \vee \psi}(E_1 \cup E_2).$$

In Figure 1 we have listed all rules introduced so far (rules (1) to (4)). In addition, the rules (5) and (6) express important logical properties.

In case an integrity constraint consists of a conjunction of several conditions, each of these conditions can be considered as a single integrity constraint valid for the same extension (rule (5)). Furthermore, we can weaken each constraint by disjunctively adding an arbitrary further condition (rule (6)). Rule (7) says that two constraints being valid for the same extension may be put together into

Let ϕ and ψ be object constraints, then the following rules hold:

$$\Theta_\phi(E) \to \Theta_{\phi \wedge \psi}(\sigma_\psi(E)) \quad (1)$$

$$\Theta_\phi(E_1 \cup E_2) \to \Theta_\phi(E_1) \quad (2)$$

$$\Theta_\phi(E_1), \Theta_\psi(E_2) \to \Theta_{\phi \wedge \psi}(E_1 \cap E_2) \quad (3)$$

$$\Theta_\phi(E_1), \Theta_\psi(E_2) \to \Theta_{\phi \vee \psi}(E_1 \cup E_2) \quad (4)$$

$$\Theta_{\psi \wedge \phi}(E) \to \Theta_\psi(E) \quad (5)$$

$$\Theta_\psi(E) \to \Theta_{\psi \vee \phi}(E) \quad (6)$$

$$\Theta_\phi(E), \Theta_\psi(E) \to \Theta_{\phi \wedge \psi}(E) \quad (7)$$

$$\to \Theta_{\mathsf{true}}(E) \quad (8)$$

Fig. 1. Overview of rules for *object constraints*

a conjunction which is still valid for that extension. For completeness, rule (8) allows to introduce the constraint true being valid for each extension.

Considering rule (7) we can see that it is only a special case of rule (3) where E_1 equals E_2. Because this is an important case often occurring during schema integration, we give an own rule for that.

In case E_1 equals E_2 we can also apply rule (4). Then, $\phi \vee \psi$ can be derived for E_1 (and E_2, resp.). From a formal point of view this result is correct, although it is too weak. In general, we are interested in obtaining the strongest constraint(s) which can be derived. Nevertheless, the rules allow us to derive weaker constraints.

Up to now, we have only dealt with object constraints. Due to the fact that the rules (5)–(7) describe general logical properties they can be used for arbitrary constraints including class constraints without modification. For the other rules we have now to see in which cases they can be applied to class constraints.

4 Applicability of the Rules to Class Constraints

The rules presented in the previous section hold for object constraints. The questions now is whether these rules are valid for class constraints, too. In this section we investigate certain kinds of class constraints (uniqueness constraints, referential integrity constraints, and constraints based on the aggregate functions max and min) which are relevant in practice.

4.1 Uniqueness Constraints

The rules (1), (2), and (3) also hold for uniqueness constraints (unique). Rule (1) means that all (selected) objects of the resulting set fulfill the uniqueness constraint besides the selection condition. Obviously, uniqueness constraints hold for each subset of a class extension for which they are originally defined. This follows immediately from the fact that an object which can be unambiguously identified in a class extension can also be unambiguously identified in each subset of this class extension. The rules (2) and (3) also hold for the same reason.

However, the following aspect has to be considered for all three rules: A uniqueness constraint which is originally defined on a superset is generally weakened by using the rules (1), (2), and (3). We will illustrate this aspect by using an example: Suppose, there is a class extension employees for which a uniqueness

constraint is defined. Further assume that this class extension can be decomposed into two base extensions project group 1 and project group 2. In this case, we can state that the uniqueness constraint holds for both base extensions. Nevertheless, the "decomposition" of the uniqueness constraint leads to a weakening because the relationship between the base extensions is not considered. Now, it is possible that in both base extensions an object can be inserted with the same key attribute values without violating one of the two derived uniqueness constraints. However, from the point of view of the original class extension the originally defined uniqueness constraint is violated.

Thus, we require a further rule which maintain the original integrity constraints. Consequently, we have to introduce the following *equivalence* rule

$$\Theta_\phi(E) \leftrightarrow \Theta_\phi(E_1 \cup E_2) \tag{9}$$

which says that a uniqueness constraint ϕ which holds for an extension E ($E \equiv E_1 \cup E_2$) also holds for the union of the both decomposed extensions E_1 and E_2 (and vice versa). The equivalence rule is not restricted to uniqueness constraints only; obviously, it holds for all kinds of constraints.

The rule (4) does not hold for uniqueness constraints on class extensions because this kind of constraints can be violated by the objects of the second class extension to be united. Thus, we cannot conclude that a uniqueness constraint holds for the united extension. This is also the case when in both class extension the same uniqueness constraint is defined (i.e. unique defined on the same attribute or attribute combination).

Let us consider the following example to illustrate why the rule (4) cannot be applied to uniqueness constraints: Suppose, there are two extensions project group 1 and project group 2. On each of these extensions there is a uniqueness constraint defined. The employees which are involved in the project group 1 are unambiguously identified by their social security number whereas the employees of project group 2 are distinguished using their names. How can these original integrity constraints be handled after unifying the corresponding class extensions? Obviously, we cannot assume that one of the original constraints holds for the whole united extension. On the other hand, these constraints cannot be skipped because of the requirement that the federated schema has to be complete in order to maintain the semantics of the original schemata to be integrated.

This problem can be solved by using *discriminants* [GCS95,AQFG95]. Discriminants can be used to maintain the relationship of the objects of the united extension with the original class extensions they stem from. For the example described above we can define a discriminant attribute group which can be used to specify integrity constraints on the united extension. For instance, the following integrity constraints can be derived in our example: All employee objects whose discriminant has the value 'project group 1' must have different social security numbers. Analogously, all employee objects whose discriminant has the value 'project group 2' must have different names.

4.2 Referential Integrity Constraints

Here, we restrict the discussion of the applicability of the rules for referential integrity constraints to constraints which refer to object references within a single

extension. In our example, there may be a referential integrity constraint on the class extension employees which defines that the attribute supervisor must refer to an existing employee object.

We state that the rule (1) does not hold for such kinds of constraints. This is due to the fact that the result of a selection operation is a subset of the original extension, i.e., some of the objects of the original extension (which do not fulfill the selection condition) are not in the resulting extension. Thus, originally valid references can now be undefined. In other words, we cannot assume that a referential integrity constraint holds for the resulting extension if potentially referenced objects are not in the resulting extension. The rules (2) and (3) do not hold for the same reason.

In contrast, the union rule (4) holds for referential integrity constraints. The reason is that all references of the original extension stay valid because all objects of this extension are in the united extension. Suppose, we unite the extensions project group 1 and project group 2 to the extension employee. Here, we assume that all references are valid because all referenced employees are in the united extension. However, we have to notice that the logical disjunction of the two original integrity constraints leads to a weakening of the constraints. As discussed before, a discriminant approach can be used to solve this problem.

4.3 Aggregate Integrity Constraints

Finally, we discuss the applicability of the rules to aggregate constraints defined by using the aggregate functions max and min. Here, we consider constraints of the form 'max(x) θ z' and 'min(x) θ z', respectively, whereby x is an attribute, z a constant value and θ is a comparison operator.

In general, the applicability of the rules to such kinds of constraints[2] depends on the comparison operator θ used for the formulation of such kinds of constraints.

For max-constraints it can be easily shown that the rules (1), (2) and (3) can only be applied if the following holds for the comparison operator: $\theta \in \{\leq, <\}$. If the maximum of an attribute x of an extension E is less (or less equal) than a certain value z, then the maximum of the attribute x of each subset of the extension E is also less (or less equal) than the value z. However, in case a max-constraint is defined with one of the following comparison operators $\{\geq, >, =, \neq\}$, then we cannot guarantee that the maximum of the resulting extension fulfills this constraint. Suppose, there is an extension employees on which the max-constraint 'max(salary) $>$ 4000' is defined. Here, it can happen that the object which make the condition valid in the original extension is not available in the resulting extension. Obviously, this is the case when, for instance, the selection condition is 'max(salary) $\leq$ 4000'.

Analogously, the rules (1), (2) and (3) can only be applied to min-constraints if the comparison operator θ is in $\{\geq, >\}$.

The rule (4) is obviously applicable to max-constraints if the comparison operator θ is in $\{\geq, >\}$ and for min-constraints if the comparison operator θ is in $\{\leq, <\}$.

[2] In the following, aggregate constraints based on the function max are called max-constraints. Analogously, aggregate constraints based on the function min are called min-constraints.

Assume, there is a constraint 'max(salary) $\geq$ 4000' defined on a class extension. Then, we can assume that this constraint also holds for union with another extension because this constraint is already fulfilled by the original extension. In this case, union means that additional objects (of a second extension) are put in this extension. Therefore, the maximum salary of the united extension can only be greater or equal than the maximum salary of the original extension.

Analogously, assume that there is a constraint 'min(salary) $\leq$ 2000' defined on a class extension. This constraint also holds for union because the minimum salary of the united extension can only be lower or equal than the minimum salary of the original extension.

However, there are two other cases where rule (4) is applicable, too. Firstly, it holds for max-constraints if the comparison operator θ is in $\{\leq, <\}$. Secondly, this rule can be applied to min-constraints if the comparison operator θ is in $\{\geq, >\}$. The reason for the applicability in both cases is that these kinds of constraints can be transformed into object constraints. The class constraint 'max(x) θ_1 z' with $\theta_1 \in \{\leq, <\}$ is equivalent to the object constraint 'x θ_1 z' and the class constraint 'min(x) θ_2 z' with $\theta_2 \in \{\geq, >\}$ corresponds to the object constraint 'x θ_2 z'. Due to the fact that the rule (4) is applicable to any object constraint we can conclude that this rule can also be applied to these special kinds of max- and min-constraints.

As discussed before, a discriminant is required in order to avoid the weakening of the constraints by applying rule (4). However, the results of the applicability of the rules to class constraints are summarized in the following table:

Rule	**unique**	**ref**	$\mathbf{max}_\theta$	$\mathbf{min}_\theta$
(1) *selection*	$\surd$	—	$\theta \in \{\leq, <\}$	$\theta \in \{\geq, >\}$
(2) *difference*	$\surd$	—	$\theta \in \{\leq, <\}$	$\theta \in \{\geq, >\}$
(3) *intersection*	$\surd$	—	$\theta \in \{\leq, <\}$	$\theta \in \{\geq, >\}$
(4) *union*	—	$\surd$	$\theta \in \{\geq, >, <, \leq\}$	$\theta \in \{\leq, <, \geq, >\}$

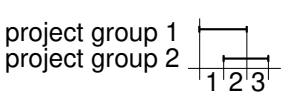
5 Example for Rule Application

In this section we demonstrate and discuss the application of our proposed rules by means of a small example. For the presentation, we distinguish two phases:

1. decomposing the original extensions into base extensions, and
2. composing these base extensions by uniting them to derived extensions.

5.1 Decomposing Extensions

The decomposition of the original class extensions is based on an extensional analysis. Such an analysis specifies in which way we have to construct the base extensions from the original extensions by applying set operations. In our example, we start with two class extensions E_1 and E_2 which contain employees working in two different projects groups. We assume that the result of the extensional analysis is as it is depicted in the following figure:



There may be employees working only for one of the two projects, and there are employees working for both projects at the same time. In consequence, the class extensions overlap.
Now, we decompose the original class extensions into three base extensions. The decomposition can be described using the set operations intersection and difference between E_1 and E_2:

$$E_1' = E_1 - E_2, \quad E_2' = E_1 \cap E_2, \quad E_3' = E_2 - E_1$$

In Figure 2 the resulting three base extensions are placed on the second level above the two original class extensions.

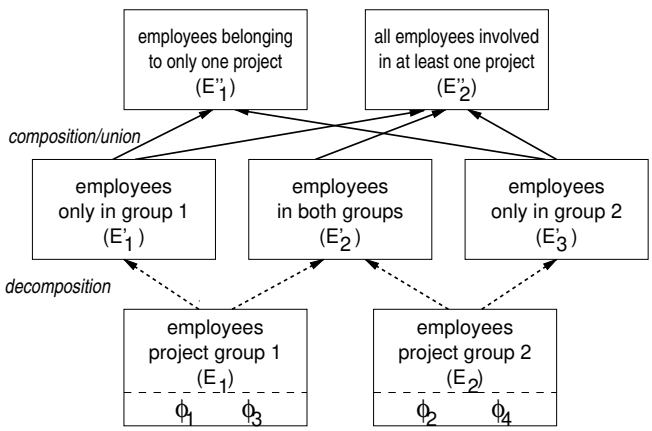


Fig. 2. Decomposing and uniting extensions of the example

Now, let us assume that we have the following integrity constraints imposed on the two original class extensions:

$$\Theta_{\phi_1}(E_1)\ with\ \phi_1 \equiv \forall e : e.\text{qualification} > 4$$
$$\Theta_{\phi_3}(E_1)\ with\ \phi_3 \equiv \forall e_1, e_2 : e_1.\text{pid} = e_2.\text{pid} \Longrightarrow e_1 = e_2$$
$$\Theta_{\phi_2}(E_2)\ with\ \phi_2 \equiv \forall e : e.\text{qualification} > 3$$
$$\Theta_{\phi_4}(E_2)\ with\ \phi_4 \equiv \exists e_1 : \forall e_2 : e_1.\text{salary_level} \geq e_2.\text{salary_level} \wedge e_1.\text{salary_level} > 3$$

ϕ_1 and ϕ_2 are object constraints whereas ϕ_3 and ϕ_4 are class constraints. ϕ_3 requires the uniqueness of values for the attribute pid (i.e., pid can be used as a key). ϕ_4 is an aggregate constraint concerning the maximum value for the attribute salary_level (we may use the short term max(salary_level)>3). In other words there must be at least one employee of extension E_2 having a higher salary level than 3.

Using the relationships between base extensions and original class extensions we can derive integrity constraints for the base extensions. For that we have to

consider each constraint and to apply our rules. The object constraints ϕ_1 and ϕ_2 do not cause any problem. By applying rule (2) and (9) we derive:

$$\Theta_{\phi_1}(E_1) \overset{\text{rule 9}}{\longrightarrow} \Theta_{\phi_1}(E_1' \cup E_2') \overset{\text{rule 2}}{\longrightarrow} \Theta_{\phi_1}(E_1')$$
$$\Theta_{\phi_1}(E_1) \overset{\text{rule 9}}{\longrightarrow} \Theta_{\phi_1}(E_2' \cup E_1') \overset{\text{rule 2}}{\longrightarrow} \Theta_{\phi_1}(E_2')$$
$$\Theta_{\phi_2}(E_2) \overset{\text{rule 9}}{\longrightarrow} \Theta_{\phi_2}(E_2' \cup E_3') \overset{\text{rule 2}}{\longrightarrow} \Theta_{\phi_2}(E_2')$$
$$\Theta_{\phi_2}(E_2) \overset{\text{rule 9}}{\longrightarrow} \Theta_{\phi_2}(E_3' \cup E_2') \overset{\text{rule 2}}{\longrightarrow} \Theta_{\phi_2}(E_3')$$

Transferring the class constraints from the original extensions to the base extensions is more complicated. For instance, the application of rule (2) to ϕ_3 leads to a weakening of the constraint. As long as we do not forget the original constraint, this is not a real problem:

$$\Theta_{\phi_3}(E_1) \overset{\text{rule 9}}{\longrightarrow} \Theta_{\phi_3}(E_1' \cup E_2') \overset{\text{rule 2}}{\longrightarrow} \Theta_{\phi_3}(E_1')$$
$$\Theta_{\phi_3}(E_1) \overset{\text{rule 9}}{\longrightarrow} \Theta_{\phi_3}(E_2' \cup E_1') \overset{\text{rule 2}}{\longrightarrow} \Theta_{\phi_3}(E_2')$$
$$\Theta_{\phi_3}(E_1) \overset{\text{rule 9}}{\longrightarrow} \Theta_{\phi_3}(E_1' \cup E_2')$$

In contrast to the uniqueness constraint ϕ_3, the max-constraint ϕ_4 cannot be transferred to any base extension. Hence, only rule (9) can be applied:

$$\Theta_{\phi_4}(E_2) \overset{\text{rule 9}}{\longrightarrow} \Theta_{\phi_4}(E_2' \cup E_3')$$

5.2 Uniting Extensions

For the decomposition of extensions the respective rules are applied to each *single* constraint. However, for the composition the application of rule (4) to each simple constraint is not useful. In order to avoid a constraint weakening, rule (4) has to be applied to *all* possible combinations of constraints and afterwards the results have to be conjunctively combined by rule (7). A simpler way to deal with constraints is to conjunctively combine the constraints of each base extension before applying other rules.

In our integration example the federated schema consists of the class extensions E_1'' and E_2''. Extension E_1'' results from uniting base extensions E_1' and E_3' and contains all employees working in only one project group whereas extension E_2'' contains all employees. The class E_1'' is a subclass of class E_2''. This federated schema is only one possible federated schema. However, it demonstrates the application of our proposed rules.

The constraints of base extensions E_1' and E_2' have to be combined by applying rule (7). On base extension E_3' only one constraint holds. Therefore, the constraints of the base extensions E_1' and E_2' only have to be combined:

$$\Theta_{\phi_1}(E_1'), \Theta_{\phi_3}(E_1') \overset{\text{rule 7}}{\longrightarrow} \Theta_{\phi_1 \wedge \phi_3}(E_1')$$
$$\Theta_{\phi_1}(E_2'), \Theta_{\phi_2}(E_2'), \Theta_{\phi_3}(E_2') \overset{\text{rule 7}}{\longrightarrow} \Theta_{\phi_1 \wedge \phi_2 \wedge \phi_3}(E_2') \leftrightarrow \Theta_{\phi_1 \wedge \phi_3}(E_2')$$

Here, constraint ϕ_2 can be omitted since it is weaker than constraint ϕ_1.

$\mathbf{E_1''}$: The rule (4) disjunctively brings together the combined constraints of the base extensions E_1' and E_3':

$$\Theta_{\phi_1 \wedge \phi_3}(E_1'), \Theta_{\phi_2}(E_3') \xrightarrow{\text{rule 4}} \Theta_{(\phi_1 \wedge \phi_3) \vee \phi_2}(E_1' \cup E_3') \leftrightarrow \Theta_{\phi_2}(E_1' \cup E_3') \xrightarrow{\text{rule 9}} \Theta_{\phi_2}(E_1'')$$

Due to the fact that $\phi_1 \Rightarrow \phi_2$ holds, the formula $(\phi_1 \wedge \phi_3) \vee \phi_2$ can be simplified to ϕ_2. Neither constraint ϕ_3 nor ϕ_4 holds on E_1''. Objects can be inserted into extension E_1'' which cannot be propagated to E_1 nor E_3 because both class constraints are violated. It is impossible to test these class constraints for extension E_1'' due to the missing base extension E_2' on the composition level. Errors from violating update operations cannot be interpreted on this level. This problem is similar to the well-known *view-update*-problem of selective views.

$\mathbf{E_2''}$: The extension E_2'' can be formed by:

$$E_2'' = E_1' \cup E_2' \cup E_3' = (E_1' \cup E_2') \cup (E_2' \cup E_3')$$

The integrity constraints of class extension $E_1' \cup E_2'$ cannot be derived directly by applying rule (4) due to the class constraint ϕ_3. This constraint has to be weakened before:

$$\Theta_{\phi_1 \wedge \phi_3}(E_1') \xrightarrow{\text{rule 5}} \Theta_{\phi_1}(E_1') \qquad\qquad \Theta_{\phi_1 \wedge \phi_3}(E_2') \xrightarrow{\text{rule 5}} \Theta_{\phi_1}(E_2')$$

The application of rule (4) produces:

$$\Theta_{\phi_1}(E_1'), \Theta_{\phi_1}(E_2') \xrightarrow{\text{rule 4}} \Theta_{\phi_1 \vee \phi_1}(E_1' \cup E_2') \leftrightarrow \Theta_{\phi_1}(E_1' \cup E_2')$$

By applying rule (7) we obtain:

$$\Theta_{\phi_1}(E_1' \cup E_2'), \Theta_{\phi_3}(E_1' \cup E_2') \xrightarrow{\text{rule 7}} \Theta_{\phi_1 \wedge \phi_3}(E_1' \cup E_2')$$

Analogously to $E_1' \cup E_2'$, we have to weaken the constraint before applying rule (4):

$$\Theta_{\phi_1 \wedge \phi_3}(E_2') \xrightarrow{\text{rule 5}} \Theta_{\phi_1}(E_2')$$
$$\Theta_{\phi_1}(E_2'), \Theta_{\phi_2}(E_3') \xrightarrow{\text{rule 4}} \Theta_{\phi_1 \vee \phi_2}(E_2' \cup E_3') \leftrightarrow \Theta_{\phi_2}(E_2' \cup E_3')$$

Due to the fact that $\phi_1 \Rightarrow \phi_2$ holds, the formula $\phi_1 \vee \phi_2$ can be simplified to ϕ_2. By applying rule (7) we obtain:

$$\Theta_{\phi_2}(E_2' \cup E_3'), \Theta_{\phi_4}(E_2' \cup E_3') \xrightarrow{\text{rule 7}} \Theta_{\phi_2 \wedge \phi_4}(E_2' \cup E_3')$$

Before the extension $E_1' \cup E_2'$ and $E_2' \cup E_3'$ can be united by applying rule (4) the constraint ϕ_3 has to be removed:

$$\Theta_{\phi_1 \wedge \phi_3}(E_1' \cup E_2') \xrightarrow{\text{rule 5}} \Theta_{\phi_1}(E_1' \cup E_2')$$

By applying rule (4) we obtain:

$$\Theta_{\phi_1}(E_1' \cup E_2'), \Theta_{\phi_2 \wedge \phi_4}(E_2' \cup E_3') \xrightarrow{\text{rule 7}} \Theta_{\phi_1 \vee (\phi_2 \wedge \phi_4)}(E_1' \cup E_2') \xrightarrow{\text{rule 9}} \Theta_{\phi_1 \vee (\phi_2 \wedge \phi_4)}(E_2'')$$

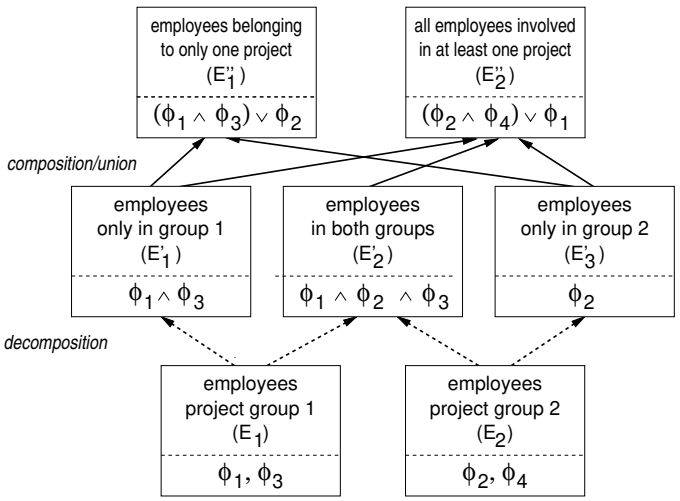


Fig. 3. Extensions with derived integrity constraints

Figure 3 shows the extensions with their integrity constraints in the unreduced form.

An interesting point is the fact that the uniqueness constraint ϕ_3 disappeared from extension E_1'' due to the disjunctive combination expressed in rule (4). The uniqueness constraint does not hold for the extension E_2'', too. The introduction of an artificial discriminant attribute avoids such an disappearance. The discriminant attribute (e.g. group) in our example expresses the information whether an employee works for project group 1 (value 1) or project group 2 (value 2) or for both of them (value 3). This information can be used within the constraints in form of an implication. The left hand side of the implication tests the discriminant attribute on a specific value and the right hand side contains the original constraint. We transform the constraints ϕ_1, ϕ_2, ϕ_3, and ϕ_4 to $\phi_1', \phi_2', \phi_3'$, and ϕ_4', respectively, and assign them to the extensions E_1', E_2', and E_3'.

$$\phi_1' \equiv \forall e : (e.\text{group} = 1 \vee e.\text{group} = 3) \Longrightarrow e.\text{qualification} > 4$$

$$\phi_2' \equiv \forall e : (e.\text{group} = 2 \vee e.\text{group} = 3) \Longrightarrow e.\text{qualification} > 3$$

$$\phi_3' \equiv \forall e_1, e_2 : ((e_1.\text{group} = 1 \vee e_1.\text{group} = 3) \wedge (e_2.\text{group} = 1 \vee e_2.\text{group} = 3)) \Longrightarrow (e_1.\text{pid} = e_2.\text{pid} \Longrightarrow e_1 = e_2)$$

$$\phi_4' \equiv \exists e_1 : \forall e_2 : ((e_1.\text{group} = 2 \vee e_1.\text{group} = 3) \wedge (e_2.\text{group} = 2 \vee e_2.\text{group} = 3)) \Longrightarrow e_1.\text{salary_level} \geq e_2.\text{salary_level} \wedge e_1.\text{salary_level} > 3$$

$$\Theta_{\phi_1' \wedge \phi_3'}(E_1'), \qquad \Theta_{\phi_1' \wedge \phi_2' \wedge \phi_3'}(E_2'), \qquad \Theta_{\phi_2'}(E_3')$$

These transformed constraints can now be conjunctively combined. The conjunctive combination stands in contrast to rule (4).

$$\Theta_{\phi_1' \wedge \phi_3'}(E_1'), \Theta_{\phi_2'}(E_3') \longrightarrow \Theta_{\phi_1' \wedge \phi_2' \wedge \phi_3'}(E_1' \cup E_3') \overset{\text{rule } 9}{\longrightarrow} \Theta_{\phi_1' \wedge \phi_2' \wedge \phi_3'}(E_1'')$$

$$\Theta_{\phi'_1 \wedge \phi'_3}(E'_1), \Theta_{\phi'_1 \wedge \phi'_2 \wedge \phi'_3}(E'_2), \Theta_{\phi'_2}(E'_3), \Theta_{\phi'_4}(E'_2 \cup E'_3)$$
$$\longrightarrow \Theta_{\phi'_1 \wedge \phi'_2 \wedge \phi'_3 \wedge \phi'_4}(E'_1 \cup E'_2 \cup E'_3) \stackrel{\text{rule 9}}{\longrightarrow} \Theta_{\phi'_1 \wedge \phi'_2 \wedge \phi'_3 \wedge \phi'_4}(E''_2)$$

Introducing discriminant attributes and respective transformation of constraints reduce weakening of constraints during schema integration (or more general during schema restructuring).

6 Conclusions

The complete treatment of integrity constraints during the schema integration is a fundamental requirement, e.g., for global integrity enforcement in federated database systems. Moreover, the extensive consideration of integrity constraints plays an important role in application scenarios such as view materialization [SJ96] or mobile databases [WSD+95].

In this paper, we presented an approach to handle integrity constraints during the schema integration. We found a small set of general rules which specify how to deal with integrity constraints during the integration process. We showed that these rules can be used for object constraints without any restrictions. For several classes of often occurring class constraints we presented the applicability of the rules. Of course, there are some logical restrictions for which specialized solutions are needed. Beside the results presented here, we have already investigated further classes of constraints, e.g. constraints with other aggregate functions.

Furthermore, we presented a systematic application of the rules. By means of a running example we sketched some critical problems such the weakening of constraints and demonstrated how to handle these kinds of problems (e.g. by using discriminants).

Our approach is not limited to schema integration purposes only, it can also be used for view derivation and arbitrary schema modification operations such as schema transformation or schema restructuring. We are currently working on the completion of the classification of integrity constraints and the applicability of our rules to further classes of constraints. Using a classification which is based on the logical structure of constraints (assuming an adequate normal form for the logical representation of constraints) seems to be a very promising approach for completing the formal part of our work.

Finally, we can state that beside our schema integration strategies presented in [SS96a,SS96b,SC97], the consideration of integrity constraints is another step towards a unified integration methodology.

References

[AQFG95] R. M. Alzahrani, M. A. Qutaishat, N. J. Fiddian, and W. A. Gray. Integrity Merging in an Object-Oriented Federated Database Environment. In C. Goble and J. Keane, editors, *Advances in Databases, Proc. of the 13th British National Conf. on Databases, BNCOD 13, Manchester, UK, July 1995*, Lecture Notes in Computer Science, Vol. 940, pages 226–248. Springer-Verlag, Berlin, 1995.

[BC86] J. Biskup and B. Convent. A Formal View Integration Method. In C. Zaniolo, editor, *Proc. of the 1986 ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C.*, pages 398–407, ACM SIGMOD Record, Vol. 15, No. 2, ACM Press, June 1986.

[BCW91] M. Bouzeghoub and I. Comyn-Wattiau. View Integration by Semantic Unification and Transformation of Data Structures. In H. Kangassalo, editor, *Entity-Relationship Approach: The Core of Conceptual Modelling, Proc. of the 9th Int. Conf. on the Entity-Relationship Approach (ER'90), Lausanne, Switzerland, October 1990*, pages 381–398. North-Holland, Amsterdam, 1991.

[CHS+97] S. Conrad, M. Höding, G. Saake, I. Schmitt, and C. Türker. Schema Integration with Integrity Constraints. In C. Small, P. Douglas, R. Johnson, P. King, and N. Martin, editors, *Advances in Databases, 15th British National Conf. on Databases, BNCOD 15, London, UK, July 1997*, Lecture Notes in Computer Science, Vol. 1271, pages 200–214. Springer-Verlag, Berlin, 1997.

[Con86] B. Convent. Unsolvable Problems Related to the View Integration Approach. In G. Ausiello and P. Atzeni, editors, *Proc. of the 1st Int. Conf. Database Theory (ICDT'86), Roma, Italy*, Lecture Notes in Computer Science, Vol. 243, pages 141–156. Springer-Verlag, Berlin, 1986.

[EJ95] L. Ekenberg and P. Johannesson. Conflictfreeness as a Basis for Schema Integration. In S. Bhalla, editor, *Information Systems and Data Management, Proc. of the 6th Conf., CISMOD'95, Bombay, India, November 1995*, Lecture Notes in Computer Science, Vol. 1006, pages 1–13. Springer-Verlag, Berlin, 1995.

[GCS95] M. Garcia-Solaco, M. Castellanos, and F. Saltor. A Semantic-Discriminated Approach to Integration in Federated Databases. In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95), May 9–12, 1995, Vienna, Austria*, pages 19–31, 1995.

[LNE89] J. A. Larson, S. B. Navathe, and R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, April 1989.

[RPG95] M. P. Reddy, B. E. Prasad, and A. Gupta. Formulating Global Integrity Constraints during Derivation of Global Schema. *Data & Knowledge Engineering*, 16(3):241–268, July 1995.

[SC97] I. Schmitt and S. Conrad. Restructuring Class Hierarchies for Schema Integration. In R. Topor and K. Tanaka, editors, *Database Systems for Advanced Applications '97, Proc. of the 5th Int. Conf., DASFAA'97, Melbourne, Australia, April 1–4, 1997*, pages 411–420, World Scientific Publishing, Singapore, 1997.

[SJ96] M. Staudt and M. Jarke. Incremental Maintenance of Externally Materialized Views. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96), Bombay, India*, pages 75–86. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[SL90] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SPD92] S. Spaccapietra, C. Parent, and Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, July 1992.

[SS96a] I. Schmitt and G. Saake. Integration of Inheritance Trees as Part of View Generation for Database Federations. In B. Thalheim, editor, *Conceptual*

Modelling — ER'96, Proc. of the 15th Int. Conf., Cottbus, Germany, October 1996, Lecture Notes in Computer Science, Vol. 1157, pages 195–210. Springer-Verlag, Berlin, 1996.

[SS96b] I. Schmitt and G. Saake. Schema Integration and View Generation by Resolving Intensional and Extensional Overlappings. In K. Yetongnon and S. Hariri, editors, *Proc. of the 9th ISCA Int. Conf. on Parallel and Distributed Computing Systems (PDCS'96), Dijon, France, September 1996*, pages 751–758. International Society for Computers and Their Application, Six Forks Road, Releigh, NC, 1996.

[VA96] M. W. W. Vermeer and P. M. G. Apers. The Role of Integrity Constraints in Database Interoperation. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96), Bombay, India*, pages 425–435. Morgan Kaufmann Publishers, San Francisco, CA, September 1996.

[WSD+95] O. Wolfson, P. Sistla, S. Dao, K. Narayanan, and R. Raj. View Maintenance in Mobile Computing. *ACM SIGMOD Record*, 24(4):22–27, 1995.