Dealing With Integrity Constraints During Schema Integration*

Stefan Conrad Ingo Schmitt Can Türker

Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg, Germany

E-mail: {conrad|schmitt|tuerker}@iti.cs.uni-magdeburg.de

— Extended Abstract —

1 Introduction

At the time being, a large number of methods have been proposed for schema integration, e.g., [SPD92, RPRG94, SS96a, SC97]. Surprisingly, most of them do not consider the integration of existing integrity constraints. Sometimes restricted kinds of constraints are considered, like in [RPG95]. As pointed out in [VA96] constraints play an important role for the interoperation of database systems. In [CHS+97] we present the results of a case study in which we investigated the treatment of constraints during the schema integration process. Thereby, a number of rules for dealing with constraints came to light. Furthermore, it turned out that these rules do not only apply in the case of schema integration but also for schema transformation and view derivation. In fact, we can use these rules even for arbitrary schema restructuring.

In this abstract we present and discuss the major rules for dealing with integrity constraints. For that, we first identify a set of elementary operations for schema integration and schema restructuring. For these operations we present our rules. By means of a small example we then demonstrate the application of these rules. Although our rules already cover a large spectrum of integrity constraints, and especially those which frequently occur in practice, there are some cases where additional rules are needed. In the conclusions, we briefly consider these cases.

2 Elementary Operations

Due to the fact that integrity constraints are usually imposed on database objects or set of objects, we have to consider not only class definitions but also their potential extensions (which we call *class extensions*). When we use the term "class extension" we do not mean the current set of objects belonging to a class in the schema. Instead we think of all potential objects belonging to that class. In that understanding the current set of objects belonging to a certain class is only a subset of the class extension.

During the schema integration process the main task is to find all relationships between the class extensions in the base schemata which are to be integrated. Two class extensions can be equal, overlapping, or disjoint. A special case of overlapping class extensions is that one is a

*This work was partly supported by the German Federal State Sachsen-Anhalt under grant number FKZ 1987A/0025 ("Federating heterogeneous data base systems and local data management components for global integrity maintenance").

subset of another one (cf. e.g. [SPD92]). As showed in [SS96a, SS96b], pairwise comparison of class extensions is not always sufficient because not all pieces of information necessary for an adequate integration can be found in that way. Instead, all class extension must be considered at the same time. The result of such an analysis can be presented in a graphical manner as depicted in Figure 1. The horizontal lines represent the class extensions of 5 classes A, B, C, D, and E, respectively. For instance, the class extensions of B and C are overlapping. Furthermore, E is a subclass D. Based on this information, a set of *base extensions* can be identified as a canonical representation. In the figure, eight base extensions have be found and numbered by 1 to 8.

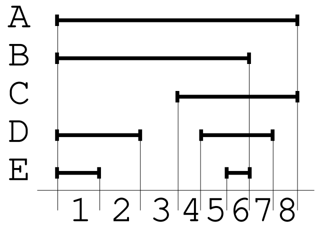

Figure 1: Extensional overlapping

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 all schema integration steps can be described in terms of the three standard set operations. 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.

For these four elementary operations we can now provide rules for treating integrity constraints when applying these operations.

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. Obviously, it is possible to distinguish several different kinds 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 for 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 resoect to integrity 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):

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

This rule can easily be transformed for direct use of set difference. We can substitute E by $E' - E''$ and E_2 by $E' - E''$. Then, we obtain the following version of this rule (because $E' = (E' - E'') \cup (E' \cap E'')$):

$$\Theta_\phi(E') \rightarrow \Theta_\phi(E' - E'').$$

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 each 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 also 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 2 we have listed all rules introduced so far (rule (1) to (4)). In addition, the rules (5) and (6) express important logical properties. In case an integrity constraints consists of a conjunction of several conditions, each of these conditions can be considered as a single integrity

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

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

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

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

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

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

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

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

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

$$E \equiv E_1 \cup E_2 : \ \Theta_\phi(E) \leftrightarrow \Theta_\phi(E_1 \cup E_2) \quad (9)$$

Figure 2: Overview of rules for *object constraints.*

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 a conjunction which is still valid for that extension. For completeness, rule (8) allows to introduce the constraint true being valid for each extension. Finally, rule (9) is an example for the fact that we may substitute each expression standing for an extension by another expression standing for the same expression (here: E may be substituted by $E_1 \cup E_2$ due to $E = E_1 \cup E_2$; of course, the other direction of substitution is here valid, too).

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), (6), (7), and (9) 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.

Applicability of the Rules to Class Constraints

The rules presented in the last section hold for object constraints. The questions now is whether these rules are valid for class constraints, too. Due to space restrictions, we briefly present first results for certain kinds of class constraints (uniqueness conditions, referential integrity constraints, and constraints based on the aggregate functions max and min[1]) without further

[1]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.

discussing details. Figure 3 summarizes the applicability of the basis rules ((1)–(4)) to class constraints.

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*	—	$\surd$	$\theta \in \{\geq, >, <, \leq\}$	$\theta \in \{\leq, <, \geq, >\}$

Figure 3: Applicability of the Rules to Class Constraints

Rule (4) does not hold for uniqueness conditions on class extensions, because this kind of constraints can be violated by the objects of the second class extension to be united. Furthermore, the rules (1), (2) und (3) are not valid for referential integrity constraints, because this kind of constraints can be violated if some objects of the original class extension are not available anymore. The applicability of the rules to `max` und `min` constraints depends on the comparison operator θ used for the formulation of such kinds of constraints. For example, the rules (1), (2) und (3) cannot be applied to the constraint `max(age) > 20`. Here, it can happen that the object which make the condition valid in the original extension is not available in the resulting extension.

4 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: (a) decomposing the original extensions into base extensions, and (b) composing these base extensions by uniting them to derived extensions.

Decomposing Extensions

The decomposition of the original class extensions is based on an extensional analysis. Such an analysis tells us 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 for employees working for two projects. We assume that the result of the extensional analysis is as it is depicted in Figure 4. There are (or 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.

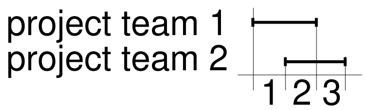

Figure 4: Extensional overlapping

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:

$$E'_1 = E_1 - E_2$$
$$E'_2 = E_1 \cap E_2$$
$$E'_3 = E_2 - E_1$$

In Figure 5 the resulting three base extensions are placed on the second level above the two original class extensions.

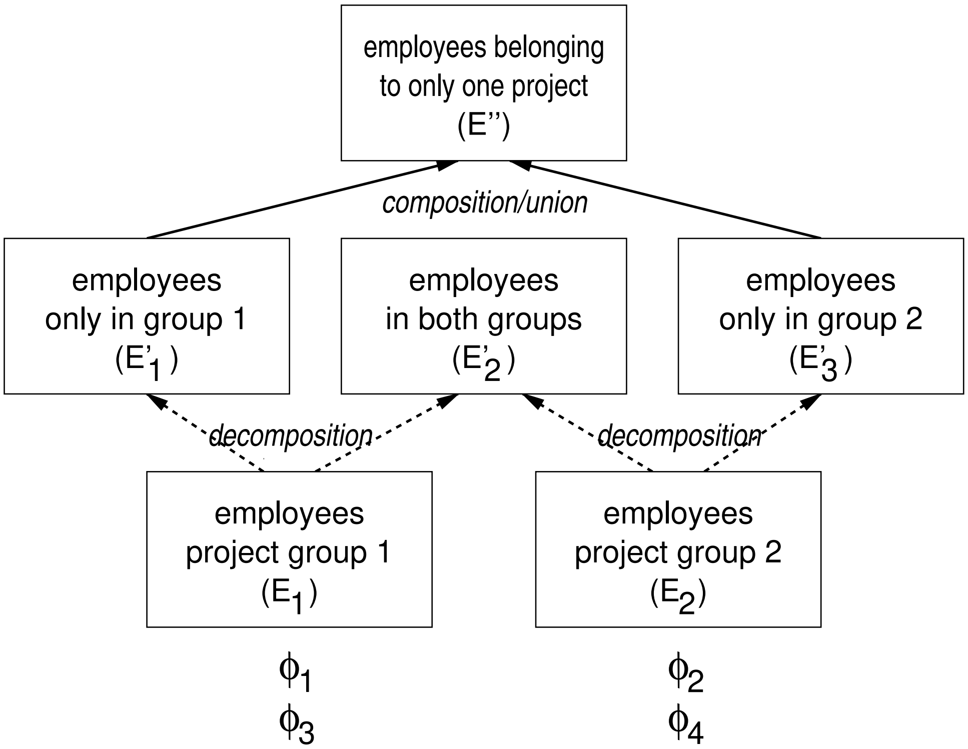

Figure 5: Decomposing and uniting extensions in the example

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

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

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 aggregation condition concerning the maximum value for the attribute salary_level (we may use the short term max(salary_level)>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, ϕ_2) do not cause any problem. By applying rule (2) and (9) we can derive:

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

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.

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

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

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

Uniting Extensions

For the decomposition of extensions the respective rules are applied to each *single* constraint. The application of rule (4) to each simple constraint is not useful. In order to avoid a constraint weakening, rule (4) have 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 rule (4).

In the introduced example, extension E'' results from uniting base extensions E_1' and E_3' and contains all employees working in only one project team. The constraints of base extensions E_1' and E_2' have to 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:

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

Constraint ϕ_2 is weaker than constraint ϕ_1. This is the reason why ϕ_2 can be omitted. The rule (4) disjunctively brings together the combined constraints of the base extensions E_1' and E_3':

$$\Theta_{\phi_1 \wedge \phi_3}(E_1'), \Theta_{\phi_2}(E_3') \stackrel{\text{rule 4}}{\longrightarrow} \Theta_{(\phi_1 \wedge \phi_3) \vee \phi_2}(E_1' \cup E_3') = \Theta_{\phi_2}(E'')$$

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''. Objects can be inserted into extension E'' 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'' 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.

An interesting point is the fact that even the weakened uniqueness constraint disappeared due to the disjunctive combination expressed in rule (4). 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 1 or project 2 or for both of them. This information can be used within the constraints in form of an implication. The left hand side of the implication tests the discriminator attribute on a specific value and the right hand side contains the original constraint.

$$\begin{aligned}
\Theta_{\phi'_1 \wedge \phi'_3}(E'_1) \quad \textit{with} \quad & \phi'_1 \equiv \forall e : e.\text{group} = 1 \Longrightarrow e.\text{qualification} > 4 \\
& \phi'_3 \equiv \forall e_1, e_2 : (e_1.\text{group} = 1 \wedge e_2.\text{group} = 1) \\
& \qquad\qquad \Longrightarrow (e_1.\text{pid} = e_2.\text{pid} \Longrightarrow e_1 = e_2) \\
\Theta_{\phi'_2}(E'_3) \quad \textit{with} \quad & \phi'_2 \equiv \forall e : e.\text{group} = 2 \Longrightarrow e.\text{qualification} > 3
\end{aligned}$$

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

$$\Theta_{\phi'_1 \wedge \phi'_3}(E'_1), \Theta_{\phi'_2}(E'_3) \longrightarrow \Theta_{\phi'_1 \wedge \phi'_2 \wedge \phi'_3}(E'_1 \cup E'_3) = \Theta_{\phi'_1 \wedge \phi'_2 \wedge \phi'_3}(E'')$$

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

5 Conclusions

In this paper, we briefly sketched an approach to handling integrity constraints during the schema integration. We found s set of rules which specify how to deal with integrity constraints during the integration process.

Our approach is not limited to schema integrity purposes, it can also be used for view derivation and arbitrary schema modification operations such as schema transformation or schema restructuring. In the future, we have to study further kinds of class constraints to provide general foundation for possible integrity constraints.

For global integrity enforcement in federated database systems [Tür96, TC97], e.g., the complete treatment of integrity constraints during the schema integration is a fundamental requirement. Moreover, the extensive consideration of integrity constraints plays an important role in application scenarios such as view materialization [SJ96, KR95] or mobile databases [WSD+95].

References

[CHS+97] S. Conrad, M. Höding, G. Saake, I. Schmitt, and C. Türker. Schema Integration with Integrity Constraints. In *15th British National Conf. on Databases, BNCOD 15, London, UK, July 1997*, Lecture Notes in Computer Science. Springer-Verlag, 1997. *To appear.*

[KR95] H. A. Kuno and E. A. Rundensteiner. Materialized Object-Oriented Views in Multi-View. In O. Bukhres, M. T. Özsu, and M.-C. Shan, editors, *RIDE-DOM'95, Proc. of the 5th Int. Workshop on Research Issues in Data Engineering: Distributed Object Management, March 6–7, 1995, Taipei, Taiwan*, pages 78–85, Los Alamitos, CA, 1995. IEEE Computer Society Press.

[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.

[RPRG94] M. P. Reddy, B. E. Prasad, P. G. Reddy, and A. Gupta. A Methodology for Integration of Heterogeneous Databases. *IEEE Transactions on Knowledge & Data Engineering*, 6(6):920–933, December 1994.

[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)*, pages 75–86. Morgan Kaufmann Publishers, 1996.

[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*, volume 1157 of *Lecture Notes in Computer Science*, pages 195–210, Berlin, 1996. Springer-Verlag.

[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*, pages 751–758, September 1996.

[TC97] C. Türker and S. Conrad. Towards Maintaining Integrity of Federated Databases. In *BIWIT'97, Proc. of the 3rd Int. Workshop on Information Technology, Bayonne, France, July 2–4, 1997*, 1997. *To appear.*

[Tür96] C. Türker. An Approach to Supporting Semantic Integrity of Federated Databases. In *Heterogeneous Information Management, Prague, Czech Republic, 4–5 November 1996, Proc. of the 10th ERCIM Database Research Group Workshop*, pages 51–60. ERCIM-96-W003, European Research Consortium for Informatics and Mathematics, 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, 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.