FUSE BY: Syntax und Semantik zur Informationsfusion in SQL

Jens Bleiholder, Felix Naumann
Humboldt-Universität zu Berlin
{bleiho,naumann}@informatik.hu-berlin.de

Abstract: Daten und Informationen heterogener Quellen können gleiche Objekte repräsentieren und dennoch sich widersprechen oder sich ergänzen. Werden solche Daten integriert, entstehen Datenkonflikte. Wir beschreiben eine einfache Ergänzung von SQL, die Daten über gleiche Objekte fusioniert. Der FUSE BY Operator vereint sich ergänzende Daten zu einem Tupel und löst gegebenenfalls Datenkonflikte. Neben der Syntax des Ausdrucks beschreiben wir seine einfache und intuitive Semantik. Schließlich werden Erweiterungen diskutiert, die es erlauben den Ausdruck um Expertenwissen zu ergänzen.

1 Informationsfusion

Integrationssysteme stellen Nutzern eine einheitliche Sicht auf mehrere Informationsquellen zur Verfügung. Die Abfrage der einzelnen Informationsquellen und das Zusammenführen der Informationen aus den Quellen übernimmt dabei das Integrationssystem. Dabei müssen auftretende Konflikte auf Schema- und Datenebene gelöst werden. Unter Informationsfusion in Integrationssystemen (auch: Data Merging) wird im Folgenden der Prozess des Zusammenführens der Daten aus den einzelnen Quellen unter Auflösung von auftretenden Konflikten verstanden. Dabei wird Objektidentität vorrausgesetzt, z.B. durch eine global eindeutige und konsistente ID.

Abbildung 1 zeigt als Beispiel zwei Tabellen, die jeweils die Daten einer Informationsquelle repräsentieren. Die Tabellen überlappen sich sowohl vertikal in den Spalten, als auch horizontal in den Personen. Desweiteren ergänzen sich die Tabellen: Spalte PKW ist nur in Tabelle Q1 enthalten, Spalte TELEFON ausschließlich in Tabelle Q2. Wie leicht zu erkennen ist, ergeben sich auf Datenebene möglicherweise Konflikte. So sind z.B. die Personen *Melanie*, *Jens* und *Christoph* in beiden Tabellen beschrieben. Es sind zwei verschiedene Konfliktarten zu unterscheiden: a) „Unsicherheiten“, also fehlende Information durch NULL Werte in den Tabellen, z.B. das Alter von *Jens* in Tabelle Q1 und b) „Widersprüche“, z.B. das Alter von *Christoph* .

Möchte man die Daten aus beiden Tabellen in eine einzige Tabelle überführen, bzw. anfragen, stellt sich die Frage, wie man mit den auftretenden Konflikten umgeht. Dieses Problem wurde erstmals in [Da83] erwähnt. Seither gibt es verschiedene Lösungsansätze, vor allem zur Konfliktvermeidung (Beseitigung von Unsicherheit). Gleichwohl gibt es kein In-

Daten aus Quelle Q1			
NAME	ALTER	STUDENT	PKW
Felix	⊥	nein	Ford
Melanie	22	ja	⊥
Jens	⊥	ja	VW
Christoph	25	ja	Citroen

Daten aus Quelle Q2			
NAME	ALTER	STUDENT	TELEFON
Melanie	⊥	ja	030/12345
Jens	27	⊥	030/54321
Christoph	24	ja	⊥
Melanie	21	nein	030/98765

Ergebnis nach Fusion von Quelle Q1 und Quelle Q2				
NAME	ALTER	STUDENT	PKW	TELEFON
Felix	⊥	nein	Ford	⊥
Melanie	22	ja	⊥	030/98765
Jens	27	ja	VW	030/54321
Christoph	25	ja	Citroen	⊥

Abbildung 1: Beispiele für Konflikte in 2 Tabellen und ihre Auflösung

tegrationssystem und keine relationale Technik, die ein Ergebnis wie in Abbildung 1 unten erzeugt. Daher schlagen wir einen neuen SQL Operator FUSE BY vor, der Konflikte nicht nur vermeidet, sondern Tabellen unter Auflösung von Konflikten zusammenführt.

2 Literatur- und Systemüberblick

Teil des neuen SQL 2003 Standards [EMK+04] ist das Merge-Statement als abkürzende Schreibweise für eine Folge von Insert-/Update-/Delete-Statements. Damit ist es möglich, eine Tabelle um Datensätze zu erweitern, wobei in der Regel bestehende Datensätze (bestehender Schlüssel) geändert und neue Datensätze (neuer Schlüssel) hinzugefügt werden. Merge fügt einer Tabelle Zeilen hinzu anstatt zwei Tabellen zu verbinden, bei der Änderung von Attributwerten stehen nur die vorhandenen Werte zur Verfügung, die Nutzung von komplexeren Konfliktlösungsfunktionen ist nicht möglich.

Der ”Match Join” Operator wird im Rahmen des Systems AURORA definiert [YÖ99]. Er lässt sich als Outer Join über das Schlüsselattribut zwischen allen Wertekombinationen aller Attribute umschreiben. Die Attributwerte sind zur Durchführung des Joins jeweils um den Schlüsselwert ergänzt. Je nach Parametrisierung des Operators werden alle Informationen bzw. Schlüssel erhalten oder nicht. Unsicherheiten werden beseitigt, Widersprüche allerdings nicht aufgelöst.

Auf dem ”Match Join” aufbauend werden in [GPZ01] der Merge (⊠) und der Prioritized Merge (◁) Operator definiert. Merge lässt sich als Vereinigung zweier Outer Joins in SQL umschreiben und zeigt, wie die SQL Funktion COALESCE zur Beseitigung von Unsicherheiten verwendet werden kann. Widersprüche werden durch Merge aber nicht aufgelöst. Durch COALESCE kann auch eine Priorisierung der Quellen angegeben werden.

In TSIMMIS [GMPQ+97] werden semistrukturierte Daten aus verschiedenen Quellen zusammengeführt. Objekte mit gleicher semantischer ID werden zusammengeführt, fehlende Informationen ergänzt, Redundanzen entfernt und Widersprüche vermieden, indem pro Attribut eine bevorzugte Quelle festgelegt wird [PAGM96]. Dies geschieht im Mediator durch Angabe von Regeln in einer Mediator-Spezifikationssprache.

Benutzerdefinierte Aggregation und Gruppierung ist ein relationaler Ansatz Daten einer

Tabelle zu fusionieren, indem zuerst Gruppen gleicher Objekte gebildet werden, die dann durch Aggregation zu einem einzigen Objekt verschmolzen werden. In FRAQL [SCS00] werden auftretende Datenkonflikte in den Gruppen durch die Verwendung von benutzerdefinierten Aggregationsfunktionen gelöst. Im Gegensatz zu FRAQL erlauben FUSE BY Anfragen die direkte Integration von Tabellen unterschiedlicher Schemata und durch intuitives default-Verhalten ist die explizite Definition von *reconciliation functions* nicht erforderlich. Sind jedoch komplexere Transformationen zwischen Daten verschiedener Schemata notwendig, bietet FRAQL eine flexible Mapping-Sprache, die unseren Ansatz komplementiert.

3 Die SQL FUSE BY Anfrage

Als Erweiterung von SQL schlagen wir die im Folgenden beschriebene FUSE BY Anfrage vor, die Tabellen unter Auflösung von Datenkonflikten zusammenführt. Sowohl in Syntax als auch in Semantik ist sie an den SQL `GROUP BY` Ausdruck angelehnt. Die wesentlichen Unterschiede sind die Erweiterung auf mehr als eine Tabelle und variable Konfliktlösung statt Aggregation oder Konfliktvermeidung.

Syntax. FUSE BY erweitert Select-Project-(Join)-Anfragen; Abbildung 2 zeigt das Syntaxdiagramm. Das Schlüsselwort `RESOLVE` markiert Spalten mit potentiellen Konflikten und kann zusätzlich eine Konfliktlösungsfunktion (*`function`*) für Werte dieser Spalte angeben. Die `FUSE BY` Komponente bestimmt (ähnlich wie `GROUP BY`), welche Objekte fusioniert werden. Mit dem Schlüsselwort `ON ORDER` kann die Reihenfolge beeinflusst werden, in der die Tupel in die Konfliktlösung eingehen.

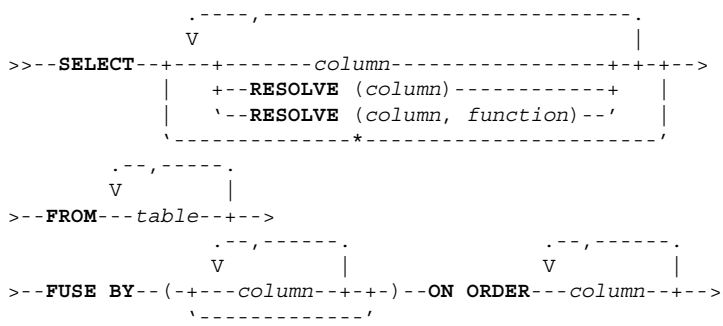

Abbildung 2: Syntaxdiagramm der FUSE BY Anfrage

Semantik. Der allgemeine Charakter der FUSE BY Anfrage ist der einer Gruppierung auf einer Outer Union von mehreren Tabellen, wobei auftretende Datenkonflikte innerhalb der Gruppen durch Konfliktlösungsfunktionen aufgelöst werden. FUSE BY Anfragen zeichnen sich durch intuitives default-Verhalten aus. Insbesondere wird, falls keine Angaben zur Gruppierung gemacht werden, die natürlichste aller Integrationen durchgeführt: Entfernung doppelter und subsumierter Tupel. Wird keine Angabe zur Konfliktlösung gemacht, werden bekannte NICHT NULL Werte NULL Werten vorgezogen. Andererseits können für jede Spalte beliebig komplexe Konfliktlösungsfunktionen angegeben werden, die als Pa-

```
SELECT *
FROM Q1
FUSE BY (A)
```
(a) Anfrage 1

```
SELECT *
FROM Q1
FUSE BY ()
```
(b) Anfrage 2

```
SELECT *
FROM Q1, Q2
FUSE BY ()
```
(c) Anfrage 3

```
SELECT Name, RESOLVE(Alter, max), RESOLVE(Student, vote), RESOLVE(Pkw),
       RESOLVE(Telefon)
FROM Q1, Q2
FUSE BY (Name) ON ORDER Q2.Alter DESC
```
(d) Anfrage 4

Abbildung 3: Beispiele für FUSE BY Anfragen

rameter den gesamten Daten- und Anfrage-Kontext erhalten. Dieser Kontext setzt sich neben den Werten selbst auch aus den dazugehörenden Tupeln, den restlichen Spaltenwerten und anderen Metadaten wie Spaltenname, Tabellenname oder Datenquelle zusammen. Konfliktlösungsfunktionen und ihre geschickte Anwendung in FUSE BY Anfragen kapseln das zur Integration benötigte Expertenwissen. Eine Erweiterung um neue Funktionen ist problemlos möglich.

Zur Durchführung einer FUSE BY Anfrage werden zuerst anhand der Angaben im FUSE BY Teil die Tupel der Ergebnistabelle bestimmt und welche davon jeweils zusammengefügt werden. Die Tupel stammen aus den Tabellen im FROM Teil, vereinigt durch Outer Union und gruppiert nach den Spalten im FUSE BY. Exakte Duplikate und subsumierte Tupel werden pro Gruppe entfernt, erst danach wird projiziert. Generell gilt, wie bei Gruppierung durch GROUP BY, dass in Spalten, die nicht in der FUSE BY Klausel stehen, Konflikte durch RESOLVE gelöst werden müssen. Wildcards (*) werden durch die Default-Funktion SQL COALESCE ersetzt: COALESCE gibt den ersten übergebenen Parameterwert der NICHT NULL ist zurück. Der so für die jeweils anderen Spalten genommene NICHT NULL Wert hängt von der Reihenfolge der Tupel ab und kann durch die ON ORDER Anweisung beeinflusst werden. Werden mehrere Tabellen angegeben, spielt zusätzlich noch die Reihenfolge der Ursprungstabellen eine Rolle.

In Anfrage 1 (Abb. 3) wird nach Werten in A gruppiert. Alle anderen Spalten der Tabelle Q1 enthalten möglicherweise Konflikte, die mittels SQL COALESCE aufgelöst werden. Somit verhält sich diese Anfrage wie eine reguläre GROUP BY Anfrage mit einer COALESCE Aggregation. Entsprechend kann auch nach mehr als einer Spalte fusioniert werden.

In Anfrage 2 (Abb. 3) wird in der FUSE BY Klausel keine Spalte genannt. Somit wird nicht gruppiert; es werden lediglich doppelte und subsumierte Tupel entfernt. Widersprüche werden nicht aufgelöst und das Ergebnis entspricht der Subsumption (in [GL94] durch $Q1\downarrow$ bezeichnet). Werden in der FROM Klausel einer FUSE BY Anfrage mehr als eine Tabelle genannt, werden diese implizit durch eine Outer Union verknüpft. Somit werden Spalten gleichen Namens zusammengeführt, fehlende Spalten werden jeweils durch NULL Werte aufgefüllt. Gegebenenfalls kann durch Umbenennung von Spalten Namensgleichheit herbeigeführt werden. Anfrage 3 führt Tabellen Q1 und Q2 zusammen, ergänzt fehlende Werte um NULL Werte und entfernt doppelte und subsumierte Tupel. Zusammen mit COALESCE als Default-Funktion entspricht dies der Ausführung des Minimum Union Operators [GL94]. Die Erweiterung auf drei und mehr Tabellen ist trivial.

Anfrage 4 (Abb. 3) liefert mit den beiden Tabellen des Beispiels aus Abbildung 1 als Input das in derselben Abbildung dargestellte Anfrageergebnis. Konflikte in der Spalte

ALTER (z.B. bei *Melanie*) werden aufgelöst, indem das höchste Alter genommen wird (`max`). Konflikte in STUDENT werden durch einen Mehrheitsentscheid aufgelöst (`vote`) und Konflikte in PKW und TELEFON löst die COALESCE-Funktion, d.h. es wird ein NICHT NULL Wert, falls vorhanden, übernommen. Beispielsweise ergibt sich die Telefonnummer von *Melanie* wegen der angegebenen Sortierung nach den Werten der Spalte ALTER in Q2 und der Reihenfolge der Quellen im FROM Teil.

4 Ausblick

Der Humboldt-Merger (HumMer) ist ein System zur Integration von relationalen und semistrukturierten Informationsquellen, das in der Arbeitsgruppe Informationsintegration der Humboldt-Universität zu Berlin entwickelt wird. Die Komponente zur Informationsfusion wird momentan durch die Implementierung des FUSE BY Operators mit Hilfe der XXL Bibliothek [dBBD+01] erstellt. Die Betrachtung, Ausführung und Optimierung von Anfragen, die neben relationalen auch Fusionsoperationen enthalten, ist dabei der nächste Schritt auf dem Weg zu einem funktionierenden Gesamtsystem.

Literatur

[Da83] Dayal, U.: Processing queries over generalization hierarchies in a multidatabase system. In: *Proc. of VLDB*. S. 342–353. 1983.

[dBBD+01] den Bercken, J. V., Blohsfeld, B., Dittrich, J.-P., Krämer, J., Schäfer, T., Schneider, M., und Seeger, B.: XXL - a library approach to supporting efficient implementations of advanced database queries. In: *Proc. of VLDB*. S. 39–48. 2001.

[EMK+04] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.-E., und Zemke, F.: SQL:2003 has been published. *SIGMOD Rec.* 33(1):119–126. 2004.

[GL94] Galindo-Legaria, C.: Outerjoins as disjunctions. In: *Proc. of SIGMOD*. S. 348–358. 1994.

[GMPQ+97] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., und Widom, J.: The TSIMMIS approach to mediation: Data models and languages. *J. Intell. Inf. Syst.* 8(2):117–132. 1997.

[GPZ01] Greco, S., Pontieri, L., und Zumpano, E.: Integrating and managing conflicting data. In: *Revised Papers from the 4th Int. Andrei Ershov Memorial Conf. on Perspectives of System Informatics*. S. 349–362. 2001.

[PAGM96] Papakonstantinou, Y., Abiteboul, S., und Garcia-Molina, H.: Object fusion in mediator systems. In: *Proc. of VLDB*. S. 413–424. 1996.

[SCS00] Sattler, K., Conrad, S., und Saake, G.: Adding Conflict Resolution Features to a Query Language for Database Federations. In: *Proc. 3rd Int. Workshop on Engineering Federated Information Systems, EFIS*. S. 41–52. 2000.

[YÖ99] Yan, L. L. und Özsu, M.: Conflict tolerant queries in AURORA. In: *Proc. of CoopIS*. S. 279. 1999.