in: Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", Kaiserslautern, März 1991, Hrsg. H.-J. Appelrath, Informatik-Fachberichte 270, Springer-Verlag, S. 502-507

Praktische Behandlung von Nullwerten -
Realisierung im Molekül-Atom-Datenmodell

Harald Schöning

Universität Kaiserslautern
Postfach 3049
6750 Kaiserslautern

Überblick

Der folgende Beitrag gibt einen Überblick über den Umgang mit Nullwerten im Molekül-Atom-Datenmodell (MAD-Modell) und dessen Implementierung PRIMA. Zunächst beschreiben wir einige der Lösungen, die im Relationenmodell für die Behandlung von Nullwerten vorgeschlagen wurden, und stellen dann den Umgang mit Nullwerten im MAD-Modell vor, wie wir ihn in PRIMA realisiert haben. Ziel ist es, den Benutzer weitgehend von den komplexen Auswirkungen mehrwertiger Logik zu befreien. Dazu diskutieren wir auch den Umgang mit Nullwerten innerhalb der komplexen Objekte des MAD-Modells, insbesondere im Zusammenhang mit Quantoren.

1. Nullwerte in Datenbanksystemen

Seit vielen Jahren beschäftigt sich die Datenbank-Forschung mit der Darstellung von unvollständiger Information im Relationenmodell. Besonders gut untersucht ist in diesem Zusammenhang die Behandlung von Attributen, deren Wert nicht bekannt ist. Hier kann man unterscheiden zwischen vollständig unbekannten Werten (Nullwerten) und ungenau bekannten Werten, deren Wert man zwar nicht angeben, aber innerhalb des zulässigen Wertebereiches doch einschränken kann (die Farbe ist rot *oder* blau) [Li79]. Im folgenden werden wir nur den Fall des vollständig unbekannten Attributwertes betrachten.

Zunächst stellt sich die Frage, wie man einen unbekannten Attributwert darstellt. Während Date [Da82] vorschlägt, einen speziellen Wert aus dem Wertebereich mit der Semantik "Nullwert" zu belegen, plädiert Codd [Co86] dafür, spezielle "Marken" zur Kennzeichnung von Nullwerten zu verwenden, die nicht zum Wertebereich des Attributes gehören. Der erste Lösungsvorschlag hat den Nachteil, daß die Darstellung von Nullwerten wertebereichsabhängig ist. Damit ist eine einheitlich Behandlung von Nullwerten durch das Datenbanksystem praktisch unmöglich, und jeder Anwendungsprogrammierer muß die Darstellung von Nullwerten für jeden Wertebereich kennen, um Nullwerte entsprechend behandeln zu können. Insbesondere kann natürlich nicht unterschieden werden, ob ein Nullwert dargestellt werden soll, oder wirklich der Wert gemeint ist, der zur Repräsentation von Nullwerten des Wertebereich ausgewählt wurde. Der zweite Ansatz erfordert einige Erweiterungen bei der Auswertung von Ausdrücken. Welches Ergebnis liefert der Vergleich eines Wertes aus einem bestimmten Wertebereich mit einem Wert, der eben nicht aus diesem Wertebereich stammt? Der erste Ansatz zur Behandlung dieser Problematik ordnet dem Vergleich mit einem Nullwert immer einen der Wahrheitswerte TRUE oder FALSE zu (vgl. [DKV89]). Abgesehen davon, daß dies die Semantik des Nullwertes nicht wiedergibt, entstehen dadurch auch Effekte, die den Umgang mit einer solchen Regelung schwierig machen. Ordnet man dem Vergleich mit einem Nullwert z.B. stets FALSE zu, so gilt nicht mehr (NOT (a=5)) ≡ (a≠5). Als Lösungsvorschlag definiert Codd [Co86] einen dritten Wahrheitswert MAYBE (quasi einen Nullwert des Wertebereichs

BOOLEAN). Er definiert, daß das Rechnen mit einem Nullwert (+, -, ...) einen Nullwert ergibt, und daß ein Vergleich (=, <, >, ...) mit einem Nullwert MAYBE liefert. Codd führt eine entsprechende dreiwertige Logik ein. Eine Wahrheitstafel für die drei Operatoren AND, OR und NOT zeigt Bild 1.1.

AND	TRUE	MAYBE	FALSE
TRUE	TRUE	MAYBE	FALSE
MAYBE	MAYBE	MAYBE	FALSE
FALSE	FALSE	FALSE	FALSE

OR	TRUE	MAYBE	FALSE
TRUE	TRUE	TRUE	TRUE
MAYBE	TRUE	MAYBE	MAYBE
FALSE	TRUE	MAYBE	FALSE

NOT	
TRUE	FALSE
MAYBE	MAYBE
FALSE	TRUE

Bild 1.1: Wahrheitstafel für dreiwertige Logik [Co86]

Nun löst aber die Einführung eines einzigen Nullwertes noch nicht alle Probleme. Betrachten wir zum Beispiel die Kundenverwaltung eines Versandhandelsunternehmens, das seinen inländischen Kunden den Einzug der Rechnungsbeträge im Lastschriftverfahren anbietet. Nehmen wir an, daß es in der Datenverwaltung des Unternehmens eine Kundenrelation und eine Geldinstituterelation gibt, und daß bei einem Kunden eingetragen ist, von welchem Geldinstitut die monatliche Zahlung abgebucht werden soll:

Kunden (Kunden_Name, ... Geldinstitut_Name);

Geldinstitute (Geldinstitut_Name, ...)

Dann kann es sein, daß ein Kunde das Lastschriftverfahren ablehnt, z.B. weil er kein Girokonto besitzt. In diesem Fall bedeutet ein Nullwert als Wert von Geldinstitut "nicht existent". Enthält die Kartei auch potentielle Kunden (etwa nach einer Kundenwerbung), so daß noch gar nicht bekannt sein kann, ob der Kunde Lastschrifteinzug wünscht, so bedeutet ein Nullwert hier "unbekannt" (*UN*). Bei Kunden im Ausland schließlich, bei denen ein Lastschrifteinzugsverfahren banktechnisch nicht möglich ist, bedeutet ein Nullwert "nicht anwendbar". Wenn wir auch in allen drei Fällen kein betroffenes Geldinstitut angeben können, so bestehen doch Unterschiede zwischen den drei Fällen. So kann z.B. der Nullwert "nicht anwendbar" normalerweise nicht geändert werden, während es bei "unbekannt" möglich ist, daß die Information noch erhoben werden kann, oder bei "nicht existent", daß der Kunde in Zukunft ein Girokonto einrichtet und Lastschrift von diesem wünscht. Stellt man die Frage, ob ein bestimmter Kunde Lastschrifteinzug über ein bestimmtes Geldinstitut wünscht, so muß man die Frage bei "nicht existent" verneinen, während man sie bei "unbekannt" nur mit MAYBE beantworten kann. Natürlich kann man sich außer den drei hier skizzierten Nullwertarten weitere vorstellen; im Rahmen dieses Beitrages sollen die eingeführten Arten jedoch nur als Verdeutlichung dienen, so daß wir es bei diesen bewenden lassen.

Führt man nun mehr als einen Nullwert ein (etwa drei mit den oben skizzierten Bedeutungen), so muß man auch eine entsprechende mehrwertige Logik wählen (eine fünfwertige in unserem Beispiel). Diese ist jedoch einem Normalbenutzer nicht zumutbar. Wir werden im folgenden eine Lösung vorstellen, die wir für unser Datenbanksystem PRIMA gewählt haben, und die den Benutzer von der Beachtung von Nullwerten möglich entlasten soll.

2. Die Bedeutung von Nullwerten für den Benutzer

Wie muß der Anwender eines Datenbanksystems seine Anfragen formulieren, wenn eine mehrwertige Logik unterstützt wird? Zur Illustration der folgenden Überlegungen stützen wir uns auf SQL, die Betrachtungen gelten aber für andere Anfragesprachen analog. Nullwerte, die bei der Projektion von Attributen auftreten, bereiten Benutzer im allgemeinen keine Verständnisschwierigkeiten, da es nur natürlich ist, wenn manche Daten unbekannt sind. Schwierig wird es jedoch, wenn Selektionsbedingungen auf Attribute angewandt werden, deren Wert undefiniert sein kann*. Üblicherweise spezifiziert der Anwender in der WHERE-Klausel eine Bedingung Q. Dabei erwartet er, daß alle Tupel, die diese Bedingung erfüllen, zum Ergebnis gehören, und nur diese. Mit anderen Worten, die Klausel *WHERE Q* ist äquivalent zu *WHERE (Q = TRUE)*. Dies kann offensichtlich bei Verwendung einer mehrwertigen Logik zu unerwarteten Effekten führen, weil nicht mehr gilt: $(Q \neq TRUE) \Rightarrow (Q = FALSE)$. So liefert eine Tautologie der zweiwertigen Logik unter Verwendung der Logik aus Bild 1.1 nicht immer TRUE: Falls a Nullwert ist, so gilt (a <=5) OR (a>5) = MAYBE [Co86].

Diese Tatsache kann den Benutzer irreleiten: Ein typisches Zugriffsmuster ist z.B. die Unterteilung der Datenbank in mehrere Klassen, z.B. die Aufteilung in Kunden, deren Auftragsvolumen eine gewissen Wert übersteigt, und andere Kunden. Üblicherweise macht sich hier der Anwender die Tatsache zunutze, daß ein Tupel entweder Q oder NOT Q erfüllt. Er würde zwei Anfragen stellen: ... WHERE Auftragsvolumen > Wert und ... WHERE Auftragsvolumen <= Wert. Dadurch würde er die Anzahlen der jeweiligen Kunden feststellen. Daß diese u.U. gar nicht der Gesamtzahl der Kunden entspricht, weil bei einigen Kunden das Auftragsvolumen undefiniert ist, entgeht einem normalen Anwender meistens. Hier helfen auch die Speziallösungen zur korrekten Auswertung von Tautologien (z.B. [Li79]) nicht, da diese nur greifen, wenn die Tautologie explizit formuliert wird. Andererseits erfüllt auch eine systemseitige Ergänzung der Klausel *WHERE Q* zu *WHERE Q* $\neq$ *FALSE* statt *WHERE Q=TRUE* nicht die Erwartungen des Benutzers. Wenn der Benutzer nämlich in unserem Beispiel wissen möchte, zu welchem Kunden eine bestimmte Kontonummer gehört, würde er bei einer Anfrage der Form "WHERE Konto = 08154711" auf jeden Fall alle Kunden erhalten, die nicht am Lastschriftverfahren teilnehmen (weil deren Kontonummer ja undefiniert ist und daher *"Konto = 08154711"* nicht zu FALSE ausgewertet werden kann).

Es bleibt also noch die Möglichkeit, daß der Benutzer bei seiner Anfrage immer explizit den erwarteten Wahrheitswert angibt. Dies wird bei einer mehrwertigen Logik sehr schnell unübersichtlich. Will man z.B. Kunden für die eigene Hausbank werben, so wird man diejenigen Kunden nicht anschreiben, von denen man sicher weiß, daß sie dort schon ein Konto haben. In diesem Fall muß man formulieren: *WHERE ((Geldinstitut = MYBANK)* = *UN) OR ((Geldinstitut = MYBANK)* = *FALSE)*, was bei einer dreiwertigen Logik äquivalent ist zu *WHERE ((Geldinstitut = MYBANK)* $\neq$ *TRUE),* in einer fünfwertigen dagegen nicht. Diese Beispiele sollten zeigen, daß es dem Durchschnittsbenutzer kaum zuzumuten ist, den Bereich der ihm doch einigermaßen vertrauten zweiwertigen Logik zu verlassen (vgl. auch [Co87]. Andererseits kann man keine zufriedenstellende allgemeine Abbildung der verschiedenen Wahrheitswerte einer mehrwertigen Logik auf eine zweiwertige Logik finden.

In der Anfragenbearbeitung in PRIMA [HMMS87] favorisieren wir daher eine explizit durch den Benutzer durchgeführte Abbildung. Obwohl diese Lösung für das MAD-Modell entwickelt wurde, das eine Erweiterung des Relationenmodells ist, läßt sie sich auf das Relationenmodell übertragen und an unserem Beispiel demonstrieren. Dem Benutzer ist im allgemeinen bekannt, welche Attribute einen Nullwert annehmen können (ein Primärschlüsselattribut z.B. im allgemeinen nicht). Für diese kann explizit auf das Vorhandensein eines Nullwertes abgefragt werden. Dabei kann er auf einen bestimmten Nullwert prüfen (also z.B. *IS_UN(Geldinstitut)*) oder allgemein auf einen Nullwert (*IS_NULL (Geldinstitut)*)**. Das obige Beispiel würde dann so formuliert: *WHERE IS_NULL (Geldinstitut) OR (Geldinstitut* $\neq$ *MYBANK)*. Diese Formulierung kann durchaus in einer mehrwertigen Logik ausgewertet werden, sie wird jedoch immer TRUE oder FALSE liefern, und somit wieder dem Modell des Benutzers von der zweiwertigen Logik entsprechen. Nun stellt sich die Frage, was passiert, wenn der Benutzer es versäumt hat, die Existenz eines Nullwertes zu beachten. In diesem Falle wird die Auswertung der Bedingung unter mehrwertiger Logik in der WHERE-Klausel wieder einen von TRUE und FALSE verschiedenen Wert liefern. Wir behandeln diesen Fall dadurch, daß es für jede Anfrage zwei Ergebnismengen gibt. Die erste enthält alle Ergebnisse, die die WHERE-Klausel erfüllen, für die also Q=TRUE gilt. Die

---

* Wir benutzen die Formulierung "ein Attribut(wert) ist undefiniert", wenn der Wert eines Attributes ein Nullwert ist.

** Wir stellen also Operatoren, die $\perp$ und T aus [Bü87] ähneln, dem Benutzer explizit zur Verfügung.

andere, die wir Menge Ω nennen, sollte normalerweise leer sein, denn sie enthält alle Tupel, für die die Auswertung von Q weder TRUE noch FALSE ergibt. Wenn diese Ergebnismenge nicht leer ist, kann der Benutzer erkennen, daß er die Existenz von Nullwerten nicht beachtet hat. Es liegt dann in der Entscheidung des Benutzers, ob er eine korrigierte Anfrage stellt, ob er die zweite Ergebnismenge ignoriert, oder ob sein Ergebnis aus beiden Mengen bestehen soll. In jedem Fall ist er aber über das Auftreten von Nullwerten informiert, so daß Fehlschlüsse, wie sie im vorigen Kapitel angedeutet wurden, nicht auftreten sollten. Nachdem wir nun allgemein den Umgang mit Nullwerten in unserem System beschrieben haben, wollen wir einige spezielle Aspekte besprechen, die das Auftreten von Nullwerten in Komplexobjekten des MAD-Modell betreffen.

3. Nullwertbehandlung im Molekül-Atom-Datenmodell

Operationen des Molekül-Atom-Datenmodells (MAD-Modells) verarbeiten Komplexobjekte, sogenannte Moleküle. Ein Molekül ist ein gerichteter Graph mit Wurzel, dessen Knoten Atome genannt werden. Diese sind mit Tupeln des relationalen Modells vergleichbar. Die Kanten des Graphen sind durch Links zwischen Atomen bestimmt. Ein Link zeigt von einem Atom aus auf $n \geq 0$ Atome eines (möglicherweise gleichen) Atomtyps. Im Gegensatz zum Relationenmodell werden (bekannte) Beziehungen zwischen Tupeln (Atomen) also explizit verwaltet und nicht über Primärschlüssel-Fremdschlüssel-Beziehungen ausgedrückt.

Im MAD-Modell ist es somit auch ohne das Vorhandensein mehrerer Arten von Nullwerten entscheidbar, ob es kein zugehöriges Tupel gibt (leerer Link) oder ob zugehörige Tupel nicht bekannt sind (Link hat undefinierten Wert). Im Relationenmodell würde, wie bereits erwähnt, in beiden Fällen der Fremdschlüssel einen undefinierten Wert haben.

Nullwerte bei Attributen und Links entstehen im MAD-Modell u.U. sehr häufig: es ist möglich, beim Einspeichern eines Atoms nur bestimmte Attributwerte zu spezifizieren. Die anderen Attribute erhalten damit automatisch einen undefinierten Wert. Ferner ist es möglich, durch Schemaänderung allen bereits existierenden Atomen eines Typs weitere Attribute hinzuzufügen. Auch diese erhalten zunächst einen undefinierten Wert. Im Rahmen dieses Kurzbeitrags ist es nicht möglich, näher auf das MAD-Modell einzugehen. Der interessierte Leser sei auf [Mi88] verwiesen.

Anfragen im MAD-Modell liefern eine Menge von Molekülen. Diese wird zum einen durch eine vorgegebene Molekülstruktur bestimmt (**FROM**-Klausel), zum anderen durch Bedingungen eingeschränkt (**WHERE**-Klausel), die bestimmte Eigenschaften der Moleküle fordern. Da ein Molekül aus mehreren Mengen jeweils gleichartiger Atome besteht, spielen Quantoren über diesen Mengen eine große Rolle bei der Formulierung von Anfragen (vgl. Beispiel 3.1). Es können verschiedene Arten von Quantoren benutzt werden, die sich jeweils auf die Atome eines Atomtyps innerhalb eines Moleküls beziehen: EXISTS und FOR_ALL als existentieller bzw. universeller Quantor, sowie EXIST EXACTLY n, EXIST AT LEAST n, EXIST AT MOST n als spezielle Quantoren. Hier stellt sich nun die Frage, wie sich diese Quantoren im Zusammenhang mit Nullwerten verhalten sollen. Nach dem vorher schon erwähnten Prinzip sollte die Auswertung der Quantoren in möglichst wenigen Fällen ein undefiniertes Ergebnis liefern. Beispiel 3.1 illustriert einige Fälle: Für Molekül m1 kann nicht entschieden werden, ob mindestens zwei C-Atome die Bedingung erfüllen; bei m2 kann dies jedoch unabhängig vom Vorhandensein eines Nullwertes entschieden werden. Der EXISTS-Quantor kann analog zu der ODER-Verknüpfung im Zusammenhang mit Nullwerten nur TRUE oder ∅* ergeben; der FOR ALL Quantor nur ∅ oder FALSE (analog zur UND-Verknüpfung, [Co86]). Zur Berechnung der quantifizierten Bedingung Qu (a) : Q(a) ermitteln wir ein Intervall I(Q)=[min(I),max(I)], in dem die Anzahl der Atome, die die Bedingung erfüllen, liegen muß: min(I) = Anzahl der Atome a mit Q(a)=TRUE; max(I) = min(I) + Anzahl der Atome a mit Q(a)=∅. Die Quantoren lassen sich dann auf dem Intervall entscheiden:

Quantor Qu	TRUE	∅	FALSE
EXISTS	min(I)>0	0=min(I)<max(I)	max(I)=0
EXIST AT LEAST n	min(I)≥n	min(I)<n≤max(I)	max(I)<n
EXIST EXACTLY n	min(I)=max(I)=n	min(I)≤n≤max(I) UND min(I)<max(I)	min(I)>n ODER max(I)<n

* ∅ kennzeichnet im folgenden einen Nullwert (auch einen boole'schen Nullwert wie MAYBE)

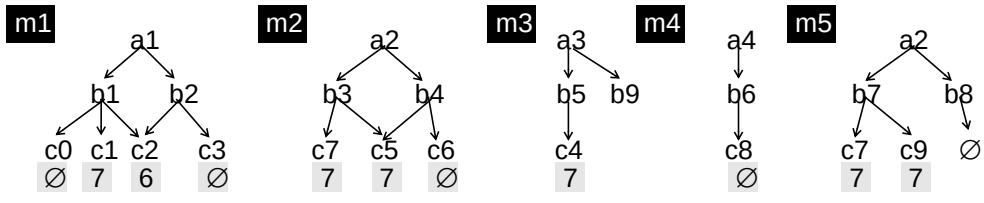
EXIST AT MOST n	max(I)≤n	min(I)≤n<max(I)	min(I)>n
FOR_ALL	min(I)=max(I)=g*	max(I)=g UND min(I)<max(I)	max(I)≠g

Auf diese Weise kann man in vielen Fällen, in denen Nullwerte in Molekülen auftreten, dennoch entscheiden, ob sich ein Molekül qualifiziert oder nicht (vgl. Beispiel 3.1). Undefinierte Links passen sich in das Schema ein. Hat ein Link auf dem Weg zu dem betrachteten Atomtyp einen undefinierten Wert (s. der Link von b8 auf C-Atome in m5 in Beispiel 3.1), so wird max(I) auf ∞ gesetzt. Damit wird ausgedrückt, daß noch beliebig viele Atome des betrachteten Typs zum Molekül gehören können. Wie das Molekül m5 in Beispiel 3.1 zeigt, läßt sich somit dasselbe Berechnungsverfahren anwenden. Molekül m3 zeigt weiterhin, daß das Verfahren auch auf Moleküle anwendbar ist, in denen keine Nullwerte auftreten.

Anfrage:

```
SELECT   ALL
FROM     A-B-C
WHERE    Qu : C.Att1=7
```

Beispielmoleküle:



Wert von C.Att1

I(Q)=[1,3] I(Q)=[2,3] I(Q)=[1,1] I(Q)=[0,1] I(Q)=[2,∞]

Die Auswertung der WHERE-Klausel ergibt für:

Qu=	TRUE	∅	FALSE
EXISTS C	m1, m2, m3, m5	m4	
EXIST EXACTLY 1 C	m3	m1, m4	m2, m5
EXIST AT LEAST 2 C	m2, m5	m1	m3, m4
FOR ALL C	m3	m2, m4, m5	m1

Beispiel 3.1: Die Auswertung von Quantoren in PRIMA unter Berücksichtigung von Nullwerten

Selbstverständlich ist diese Auswertung für Quantoren mit der im vorigen Kapitel vorgestellten allgemeinen Nullwertbehandlung kombiniert. Sie erhöht aber die Wahrscheinlichkeit, daß bei einem Auftreten von Nullwerten, das der Benutzer nicht berücksichtigt hat, trotzdem eine leere Ergebnismenge Ω entsteht.

* g ist die Gesamtzahl der betrachteten Atome

4. Zusammenfassung und Ausblick

Der Umgang mit mehrwertiger Logik ist dem normalen Benutzer nicht zuzumuten. Daher sind Mechanismen erforderlich, die es dem Benutzer so weit wie möglich erlauben, in der ihm bekannten Welt der zweiwertigen Logik zu denken. Zu diesem Zweck haben wir untersucht, wann quantifizierte Ausdrücke über Komplexobjekten des MAD-Modells (Molekülen) trotz des Auftretens von Nullwerten zu TRUE oder FALSE ausgewertet werden können. Ferner erlauben wir es dem Benutzer, eine anfrage- und attributspezifische Abbildung von Nullwerten auf einen der Wahrheitswerte TRUE oder FALSE zu definieren. Tritt wegen unvollständiger Definition dieser Abbildung ein anderer Wahrheitswert als Wert bei der Auswertung der Bedingung in der WHERE-Klausel einer Anfrage auf, so werden alle entsprechenden Moleküle in einer zusätzlichen Ergebnismenge Ω zusammengefaßt. Der Benutzer kann dann die Abbildungsvorschrift vervollständigen, indem er entweder eine neue Anfrage stellt oder die Bedingung für die Elemente von Ω selbst auswertet.

Ein in der Literatur nach meinem Wissen noch nicht behandeltes Thema ist die Aufnahme von Nullwerten in den Wert eines mehrwertigen Attributes, also z.B. das Zulassen einer Liste {1,4,∅}. Hierbei kann je nach Interpretation ein Nullwert für *ein* unbekanntes Listenelement oder für *eine unbekannte Anzahl* unbekannter Listenelemente stehen; in PRIMA verwenden wir die erste Interpretation, so daß sinnvollerweise auch mehrere Nullwerte in dem Wert eines mehrwertigen Attributes auftreten können. Solche Listen entstehen in PRIMA z.B. durch die Aggregation eines Attributwertes über den Atomen eines Moleküls. Sie lassen sich ebenfalls mit der eingeführten Intervallmethode problemlos behandeln. Analog zu den Links ist für die oben angeführte Liste in der ersten Interpretation das Intervall [2,3].

Die Auswertung von *Integritätsbedingungen* im Zusammenhang mit Nullwerten ist ein komplexes Thema. In PRIMA werden Kardinalitätsrestriktionen mengenwertiger Attribute ignoriert, wenn das Attribut einen Nullwert hat. Es gibt aber die Möglichkeit zu verbieten, daß ein Attribut einen Nullwert annimmt. Für die Überprüfung komplexer Integritätsbedingungen erscheinen diese Maßnahmen nicht als ausreichend.

Referenzen

Bü87 von Bültzingsloewen, G.: Translating and Optimizing SQL Queries Having Aggregates, in: Proc. VLDB 1987, Brighton, UK, S. 235-243.

Co86 Missing Information (Applicable and Inapplicable) in Relational Databases, in: ACM SIGMOD RECORD, Vol. 15, No. 4, 1986, pp. 53-78.

Co87 Codd, E.F.: More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable Information), in: SIGMOD RECORD, Vol. 16, No. 1, 1987, pp. 42-50.

Da82 Date, C.J.: Null Values in Database Management, Proc. 2nd British National Conf. on Databases, July 1982.

DKV89 Danforth, S., Khoshafian, S., Valduriez, P.: FAD - A Database Programming Language, Rev. 3, MCC Technical Report ACA-ST-151-85, Rev. 3, January 1989.

HMMS87 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. VLDB 1987, Brighton, UK, S. 433-442.

Li79 Lipski, W.: On Semantic Issues Connected with Incomplete Information Databases, in: ACM TODS, Vol. 4, No. 3, 1979, pp. 262-296.

Mi88 Mitschang, B.: Ein Molekül-Atom-Datenmodell für Non-Standard-Anwendungen - Anwendungsanalyse, Datenmodellentwurf und Implementierungskonzepte, Springer-Verlag Berlin Heidelberg, 1988.