Inkrementelle Integritätsprüfung und Sichtenaktualisierung in SQL

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Birgit Pieper
aus
Hamm

Bonn
2001

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Referent: Univ.-Prof. Dr. rer. nat. Rainer Manthey

2. Referent: Univ.-Prof. Dr. rer. nat. Armin B. Cremers

Tag der Promotion:

Für Martin
und meine Eltern

Kurzfassung

Zentrales Problem der Implementierung von Integritätsbedingungen und materialisierten Sichten ist die effiziente Reaktion auf Basisfaktenänderungen. Ändert sich ein Basisfakt durch Einfügung, Löschung oder Modifikation, so müssen die Integritätsbedingungen geprüft und die abgeleiteten und gespeicherten Fakten materialisierter Sichten aktualisiert werden. Da die Dauer der Integritätsprüfung und Sichtenaktualisierung unmittelbar die Ausführungsdauer einer Transaktion beeinflußt, ist eine effiziente Durchführung von entscheidender Bedeutung.

Effiziente Verfahren zur Integritätsprüfung und Aktualisierung materialisierter Sichten gehören seit mehr als 15 Jahren zu den wichtigsten Problemen der Forschungen zu deduktiven Datenbanken. Die im Kontext von Datalog und Relationaler Algebra entwickelten inkrementellen Verfahren haben jedoch bislang kaum Anwendung in SQL gefunden. Dies zeigt sich an den funktional sehr beschränkten Implementierungen beider Konzepte in SQL-basierten Datenbanksystemen kommerzieller Hersteller. Zentrale Idee inkrementeller Ansätze ist, daß Integritätsprüfung und Sichtenaktualisierung nur für die aktuell geänderten Fakten durchgeführt werden. Zu diesem Zweck werden im Rahmen eines Änderungspropagierungsprozesses die durch Basisfaktenänderungen induzierten Änderungen abgeleiteter Fakten ermittelt.

Da bei den bisherigen inkrementellen Verfahren SQL-spezifische Konzepte wie das SQL-Transaktions- und Integritätskonzept nicht berücksichtigt wurden, ist eine Anwendung dieser Verfahren in SQL nicht direkt möglich. Aus diesem Grund werden im Rahmen der vorgelegten Dissertation inkrementelle Verfahren zur effizienten Integritätsprüfung und Sichtenaktualisierung im Kontext von SQL entwickelt. Bei den Verfahrensentwürfen werden SQL-spezifische Systemeigenschaften unverändert berücksichtigt. Die Implementierung dieser Verfahren in SQL-basierten, kommerziellen Datenbanksystemen würde diese Systeme funktional um leistungsfähige Komponenten zur Integritätsprüfung und zur Simulation und Aktualisierung materialisierter Sichten erweitern.

Als Integritätskonzept wird das des SQL-Standards zugrunde gelegt. Da für materialisierte Sichten Vorgaben durch den SQL-Standard fehlen und die bisherigen Implementierungen in kommerziellen Datenbanksystemen funktional sehr beschränkt sind, wird zu diesem Zweck ein leistungsfähiges Konzept definiert. Für die Implementierungen der Verfahren in kommerziellen Datenbanksystemen werden Lösungen basierend auf SQL-Triggern und einem Präprozessor vorgeschlagen. Die drei zentralen Funktionen der SQL-basierten inkrementellen Verfahren werden jeweils von Propagierungs-, Integritätsprüfungs- und Sichtenaktualisierungstriggern realisiert. Zu den Aufgaben des Präprozessors zählen die Abhängigkeits- und Relevanzanalysen wie auch die Generierung der Prozeßtrigger und aller weiteren Datenbankobjekten, die für die Verfahrensausführung erforderlich sind.

Danksagung

Mein ganz besonderer Dank gilt Prof. Rainer Manthey für seine Bereitschaft, eine externe Promotion zu betreuen, für sein in all den Jahren großes Interesse an dem bearbeiteten Thema und insbesondere für die vielen substantiellen Diskussionen, die wesentlich zum Gelingen dieser Arbeit beigetragen haben. Prof. Armin B. Cremers danke ich sehr für sein Engagement als Zweitkorrektor dieser Dissertation. Großen Dank schulde ich auch Andreas Behrend für seine hilfreichen und insbesondere detaillierten Anmerkungen, die die Struktur der Arbeit grundlegend beeinflußt haben. Bei meinen Korrekturleserinnen Anke Pieper und Elisabeth Henning bedanke ich mich ganz herzlich für ihre Unterstützung und für ihre zahlreichen stilistischen Anmerkungen.

Eine Promotion neben meiner beruflichen Tätigkeit durchzuführen, war mir nur aufgrund des Verständnisses meines Arbeitgebers Opitz Consulting möglich. Es ist insbesondere den flexiblen Arbeitszeitmodellen zu verdanken, daß ich ausreichend Zeit gefunden habe, mich auf das Thema der Dissertation zu konzentrieren. Bei meinen Arbeitskolleginnen und -kollegen bedanke ich mich herzlich für die bereitwillige Erledigung meiner Aufgaben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Zur Erweiterung der Funktionalität von Datenbanksystemen (DB-Systemen, DBS) werden seit den 70er Jahren drei Regelkonzepte mit unterschiedlichen Zielsetzungen diskutiert.

- Normative Regeln (Integritätsbedingungen):
  Mittels Integritätsbedingungen können Konsistenzanforderungen an gespeicherte und abgeleitete Fakten einer Datenbank (DB) deklarativ formuliert werden.
- Deduktive Regeln (Sichten):
  Für die Ableitung neuen Wissens aus einer gegebenen Faktenmenge können virtuelle oder materialisierte Sichten definiert werden (deduktive DB).
- Aktive Regeln (Trigger):
  Trigger werden automatisch ausgeführt, wenn das zugehörige Ereignis eintritt (reaktive Systeme).

Diese drei Konzepte sind insofern orthogonal zueinander, als daß sie unabhängig von einander definiert und angewendet werden können. Für die Entwicklung anspruchsvoller DB-Anwendungen sind Implementierungen dieser drei Konzepte gleichermaßen relevant. Die Idee zur Entwicklung zentraler DB-Anwendungen ist motiviert durch Probleme, die in der Praxis im Rahmen des Software-Engineerings bei der Wiederverwendung von Software-Komponenten auftreten. Aufgrund der Praxisrelevanz dieser Konzepte und da bei kommerziellen Anwendungen vorwiegend SQL-basierte relationale DB-Systeme eingesetzt werden, interessiert vor allem, welche Beachtung sie im neuen SQL3-Standard und in SQL-basierten DB-Systemen kommerzieller Hersteller gefunden haben.

Trotz offensichtlicher Praxisrelevanz haben diese Konzepte jedoch nur sehr unterschiedliche Aufnahme in SQL gefunden. So sind die Integritätsbedingungen des SQL2-Standards bis heute nur ansatzweise in kommerziellen DB-Systemen realisiert worden. Funktional sehr beschränkte Konzepte materialisierter Sichten sind hingegen auch ohne Vorgaben durch den Standard entstanden. Bei den Triggern verhält es sich ähnlich. Nachdem bereits vor einigen Jahren viele DBS-Hersteller Triggerkomponenten realisiert haben, sind jetzt in SQL3 konzeptionelle Vorgaben definiert worden.

Zur Begründung der nur zögerlichen Implementierungen von Integritätsbedingungen und materialisierten Sichten wird oft mangelnde Effizienz der Integritätsprüfung und Sichtenaktualisierung genannt. Da die bisherigen Entwicklungen effizienter Verfahren für die

beiden Aufgaben im Kontext von Datalog und Relationaler Algebra erfolgt sind, liegt die Vermutung mangelnder Kenntnisse bezüglich dieser effizienten Verfahren bzw. ihrer Anwendung in SQL nahe. Aus diesem Grund wird im Rahmen dieser Dissertation die Anwendbarkeit der bisher aus anderem Kontext bekannten Verfahren in SQL diskutiert und SQL-basierte Verfahren zur effizienten Integritätsprüfung und Sichtenaktualisierung entwickelt. Da für die Verfahren eine auf SQL3-Triggern basierende Implementierung vorgeschlagen wird, wird zudem die Eignung der SQL3-Trigger zur Simulation dieser beiden 'klassischen' Anwendungsfälle im SQL-Kontext analysiert.

Problemstellung

Seit Mitte der 80er Jahre wurden eine Vielzahl inkrementeller Verfahren zur effizienten Integritätsprüfung und Sichtenaktualisierung im Kontext von Datalog ([Dec86], [LST86], [KSS87], [BDM88], [DW89], [Kue91], [Oli91], [BMM92], [GMS93], [CW94], [GrMa94], [Man94], [Grie97] u.v.m.) und Relationaler Algebra (RA) ([QW91], [GL95], [CGLMT96], [CKLMR97], [BDDF+98], [DS00], [SBLC00] u.v.m.) entwickelt. Die Grundlagen für die inkrementelle Evaluierung von Integritätsbedingungen haben Nicolas [Nic82] und Bernstein/Blaustein [BBC80] in ihren einflußreichen Arbeiten mit der Idee spezialisierter Bedingungen gelegt. Die Grundidee dieser inkrementellen Verfahren ist die Beschränkung der Integritätsprüfung bzw. Sichtenaktualisierung auf die aktuell geänderten Fakten (Δ-Fakten). Zu diesem Zweck werden die durch Änderungen von Basisfakten auf Sichten induzierten Änderungen zunächst separat abgeleitet (Änderungspropagierung) und anschließend geprüft bzw. auf materialisierten Sichten angewendet. Induzierte Änderungen sind aber nicht nur für Integritätsprüfung und Sichtenaktualisierung relevant sondern können auch als feuernde Ereignisse für Trigger oder zur Information der Anwender über die Konsequenzen von Basisfaktenänderungen von Interesse sein. Da der Propagierungsprozeß unabhängig von der weiteren Verwendung der Δ-Fakten ist, können Synergieeffekte einer einmaligen Propagierung und einer mehrfachen Verwendung der Δ-Fakten genutzt werden.

Viele Autoren verweisen zwar auf die Relevanz dieses Themas auch für SQL-basierte DB-Systeme, explizit mit SQL-Integritätsbedingungen und -Sichten setzen sich jedoch nur Ceri/Widom in ihren Arbeiten [CW90], [CW91] und Decker in [Dec01] auseinander. Auch Bello et.al. haben in [BDDF+98] die Ergebnisse ihrer Implementierung materialisierter Sichten im Oracle-DB-System im Kontext Relationaler Algebra veröffentlicht. Allen bislang publizierten Arbeiten ist gemeinsam, daß das SQL-Transaktions- und -Integritätskonzept nicht berücksichtigt wird. Im Kontext von Datalog und Relationaler Algebra ist eine Transaktion meist eine Menge von Änderungsanweisungen, die erst zum Ende einer Transaktion ausgeführt wird. Dies gilt auch für die Integritätsprüfung und Sichtenaktualisierung. Eine SQL-Transaktion ist jedoch eine Sequenz von Basisfaktenänderungen, deren Ausführung eine Sequenz von Zwischenzuständen erzeugt. Demgemäß sind zwei Integritätsprüfungszeitpunkte definiert: unmittelbar nach einer Basisfaktenänderung (IMMEDIATE) und zum Transaktionsende (DEFERRED). Die Konsistenzanforderung in SQL bezieht sich nicht nur auf den Endzustand sondern gilt auch für die Zwischenzustände.

In SQL ist zwar ein leistungsfähiges Konzept statischer Integritätsbedingungen vorgesehen, das aber nur sehr beschränkt in DB-Systemen führender, kommerzieller Hersteller implementiert worden (IBM [IBM00], Oracle [Ora99], u.v.m.) ist, so daß die Bedingungen

weder multirelational noch für Sichten definiert werden können. Zudem scheinen Unklarheiten hinsichtlich der Vorgaben des SQL-Standards zu bestehen. In der Sekundärliteratur sind dazu vielfach widersprüchliche Aussagen zu finden.

Welche Relevanz Anwender von DB-Systemen der Speicherung abgeleiteter Fakten beimessen, zeigt sich an den Implementierungen, die z.B. bei IBM (DB2 7.x) und Oracle (8i) zu finden sind. Daß ein Konzept zur Speicherung abgeleiteter Fakten in SQL3 fehlt, hat dazu geführt, daß die verschiedenen Hersteller Verfahren mit sehr unterschiedlichen Zielsetzungen realisiert haben. Beim Oracle-DB-System wird als Anwendungsgebiet für 'materialized views' Data Warehouse-Anwendungen gesehen, so daß für die Aktualisierung drei Zeitpunkte zum Ende (ON COMMIT) bzw. nach Beendigung einer Transaktion (ON DEMAND) und 'periodisch' zulässig sind. Bei dem DB2-System von IBM wird als zentrale Anwendung für 'summary tables' die performante Ausführung von Zugriffen auf abgeleitete Fakten bei Anfragen gesehen, und somit ist u.a. ein Aktualisierungszeitpunkt während der laufenden Transaktion (IMMEDIATE) zulässig. Diese Implementierungen weisen jedoch eine Vielzahl von Restriktionen auf, so z.B. eine sehr eingeschränkte Ausdruckskraft der Sichtendefinitionen, da u.a. keine Rekursion zulässig ist. Ebenfalls stark eingeschränkt sind die Möglichkeiten, Hierarchien materialisierter Sichten zu definieren. Bei DB2 ist dies z.B. gar nicht möglich. Zudem ist die Definition von Integritätsbedingungen auf (materialisierten) Sichten nicht zulässig. Sichten und Integritätsbedingungen sind zwei orthogonale Konzepte. Integritätsbedingungen auch für Sichten zu definieren, ist oft intuitiver als sie auf Basisrelationen zu transformieren. Eine solche Transformation hat die wiederholte Spezifikation der Sichtendefinition in den Integritätsbedingungen zur Folge. Diese Technik der Auffaltung der Definition ist zudem bei rekursiven Sichten (SQL3) nicht anwendbar.

Aus dem im Dezember 1999 veröffentlichten Standard SQL3 ergeben sich neue, noch nicht diskutierte Anforderungen. Neu gegenüber den Arbeiten von Ceri/Widom sind insbesondere rekursive Sichten und temporäre Hilfssichten sowie ein Konzept für Trigger. SQL-Trigger sind in diesem Zusammenhang von Interesse, weil das in dieser Dissertation entwickelte Verfahren auf ihnen basiert. Zwar sind bereits Implementierungen mittels aktiver Regeln von Ceri/Widom ([CW90], [CW91], [CW94]: Starburst) und Griefahn ([Grie97]: ActLog) vorgestellt worden, doch verfügen Starburst- und ActLog-Trigger über Ausführungsmodelle, die sich von dem der SQL-Trigger stark unterscheiden.

Ziel und Lösungsansatz

Das zentrale Ziel dieser Dissertation ist die funktionale Erweiterung kommerzieller SQL-basierter DB-Systeme um ein leistungsfähiges Integritätskonzept und um ein leistungsfähiges Konzept materialisierter und virtueller Sichten. Die Verfahren zur Integritätsprüfung und Sichtenaktualisierung werden unter der Zielsetzung einer möglichst effizienten Ausführung entwickelt.

Für die Integritätsprüfung wird das SQL2-Integritätskonzept umgesetzt. Hinsichtlich materialisierter Sichten muß der SQL-Standard um ein adäquates Konzept erweitert werden. Damit entgegen den bisherigen Lösungen die hier entwickelten Konzepte und Verfahren auch den SQL-spezifischen Anforderungen genügen, werden die Vorgaben des SQL3-Standards unverändert zugrunde gelegt. Dies gilt insbesondere für das SQL-Transaktions-

und Integritätskonzept mit den beiden Prüfungszeitpunkten IMMEDIATE und DEFERRED.

Damit die SQL-basierten Verfahren auch in bereits existierenden kommerziellen DB-Systemen angewendet werden können, ist eine Lösung auf der Grundlage eines Präprozessors und SQL-Triggern gewählt worden. Zentrale Aufgabe des Präprozessors ist die Analyse der Abhängigkeiten zwischen Basisrelationen, Sichten und Integritätsbedingungen. Auf der Basis dieser Ergebnisse ermittelt der Präprozessor, welche Änderungen relevant sind, und generiert die für die inkrementellen Verfahren erforderlichen DB-Objekte. Zu diesen DB-Objekten gehören u.a. Δ-Basisrelationen für die Speicherung der Δ-Fakten sowie Propagierungs-, Sichtenaktualisierungs- und Integritätsprüfungstrigger. Zur Laufzeit werden dann zum IMMEDIATE- wie auch zum DEFERRED-Prozeßzeitpunkt jeweils die hier entwickelten inkrementellen Verfahren als 'bottom up'-Fixpunktprozesse ausgeführt. Die Ausführungssteuerung obliegt den Prozeßtriggern. Die Grundidee der 'bottom up' ausgeführten Änderungspropagierung mit Integritätsprüfung und Sichtenaktualisierung ist aus dem Kontext von Datalog und Relationaler Algebra nach SQL übertragen worden. Präprozessor-Lösungen sind u.a. bei den Ansätzen von Ceri/Widom und Griefahn zu finden, ebenso wie ein auf Triggern basierendes Verfahren. Die hier entwickelten Verfahren unterscheiden sich von den anderen Lösungen jedoch vor allem durch die Berücksichtigung SQL-spezifischer Anforderungen und die Verwendung von SQL-Triggern.

Resultate

Zentrale Ergebnisse der vorliegenden Dissertation sind auf SQL-Triggern basierende Verfahren zur inkrementellen Änderungspropagierung mit Integritätsprüfung und Sichtenaktualisierung in SQL. Anders als bei den bisherigen Verfahren (auch von Ceri/Widom in [CW90], [CW91]) ist beim Entwurf dieser Verfahren das SQL-Transaktions- und Integritätskonzept zugrunde gelegt worden. Als Ergebnisse des Entwurfsprozesses für diese SQL-basierten Verfahren können weitere wichtige Resultate zusammengefaßt werden.

- Da für die Beurteilung der Anwendbarkeit bereits entwickelter, inkrementeller Verfahren ein grundlegendes Verständnis der Übereinstimmungen und Unterschiede zwischen Datalog und SQL erforderlich sind, werden Transformationsvorschriften definiert, mit denen Ausdrücke der beiden relationalen DB-Sprachen ineinander überführt werden können. Bislang sind für SQL nur Vorschriften zur Transformation in Ausdrücke der Relationalen Algebra ([CG85], [CGT90]) spezifiziert worden. Für Datalog-Regeln sind nur einzelne Vorschriften für Transformationen ins Bereichskalkül (DRC) definiert worden.

- Die SQL-basierten Verfahren stellen jeweils eine Art Basisverfahren dar, welche die grundlegenden Optimierungsideen inkrementeller Ansätze berücksichtigen. Weitere Lösungen für teilweise sehr spezielle Systemeigenschaften, die bereits in Datalog und Relationaler Algebra formuliert wurden, können ohne wesentliche Einschränkungen und teilweise nur unter Anpassung der Syntax integriert werden.

- Die SQL-spezifischen Anforderungen ermöglichen neue Optimierungsansätze für effiziente, inkrementelle Verfahren, wie z.B. eine zu den beiden Verarbeitungszeitpunkten verteilt ausgeführte Propagierung.

- Ebenfalls mit dem Ziel weiterer Effizienzsteigerungen bei der Verfahrensausführung werden grundlegende Regeln zur Propagierung von Modifikationen in SQL definiert.

Diese Regeln gehen weit über den von Olivé et.al. ([UO92]) im Datalog-Kontext formulierten Spezialfall der unveränderten Schlüsselattribute von Relationen hinaus.

Vor dem Hintergrund, daß Trigger allgemein als ein 'klassisches' Werkzeug zur Simulation von Integritätsprüfungen gelten, kann als Ergebnis des hier vorgestellten auf SQL-Triggern basierenden Verfahrens ein widersprüchliches Fazit gezogen werden.

- Trotz der sehr weitreichenden Unterstützung der Prozeßausführung durch die SQL-Trigger treten Probleme auf, die aus konzeptionellen Diskrepanzen zwischen dem SQL-Integritäts- und dem SQL-Triggerkonzept resultieren. Andere Triggerkonzepte, die bereits zur Implementierung inkrementeller Verfahren verwendet wurden, wie z.B. Starburst-Trigger in [CW90] und ActLog-Trigger in [Grie97] sind zur Implementierung der SQL-basierten Verfahren nicht geeignet.

Gliederung der Arbeit

In **Kapitel 2** werden die sprachlichen Grundlagen definiert soweit sie für die Diskussion bereits entwickelter, Datalog-basierter Verfahren wie auch für die Entwicklung eines SQL-basierten Verfahrens erforderlich sind. Neben der quasi standardisierten Datalog-Regelsprache werden eine Änderungssprache und ein Transaktionskonzept für Datalog definiert. Die Anfrage- und Änderungssprache von SQL3 wird in einer syntaktisch und semantisch eingeschränkten Form mit einer Basisfunktionalität definiert (Basis-SQL). Damit ein ausführlicher Vergleich verschiedener triggerbasierter Implementierungsansätze möglich ist, wird neben SQL-basierten Triggern auch ein aktives Regelkonzept für Datalog erörtert.

Um die Diskussionen im Kontext von SQL3 anschaulich führen zu können, und, um die Analogien zu Datalog zu veranschaulichen, wird in **Kapitel 3** ein syntaktisch normalisiertes NSQL definiert. Da bislang Vorschriften für eine direkte Transformation von SQL-Ausdrücken nach Datalog und umgekehrt fehlen, werden Funktionen definiert, die NSQL-Ausdrücke auf Datalog-Ausdrücke abbilden und umgekehrt.

Bei der Übersicht über die bislang in Datalog und Relationaler Algebra diskutierten inkrementellen Verfahren werden in **Kapitel 4**, anders als vielfach üblich, die verschiedenen Lösungsansätze nicht in ihrer Gesamtheit vorgestellt. Vielmehr werden die zentralen Ideen erörtert und ein generisches Verfahren vorgestellt, das grundlegenden Anforderungen wie z.B. der Bearbeitung stratifizierbarer Regelmengen und statischer Integritätsbedingungen genügt. Für die Basis-Eigenschaften eines solchen Verfahrens werden im weiteren alternative, bereits publizierte Lösungen aufgezeigt. Abschließend werden weitere Verfahren vorgestellt, die gegenüber dem generischen Verfahren funktional z.B. um Aggregation und die Propagierung von Modifikationen erweitert sind.

Bevor in **Kapitel 5** inkrementelle Verfahren im SQL-Kontext entwickelt werden, werden alternative Implementierungsstrategien ausführlich diskutiert. Für ein Änderungspropagierungsverfahren werden die SQL-spezifischen Anforderungen und Optimierungsmöglichkeiten erörtert und daraus resultierend ein SQL-Propagierungskonzept formuliert, für das dann ein auf SQL-Triggern basierender Implementierungsansatz entwickelt wird. Ausgehend von diesem Propagierungsverfahren werden für die verschiedenen Aufgabenstellungen der Integritätsprüfung, der Sichtenaktualisierung und für beide Aufgaben zusam-

men separate Lösungen entwickelt, so daß Strategien für effizientere Ausführungen angepaßt an die einzelnen Aufgaben vorgestellt werden können. Anhand eines Beispiels werden die Aufgaben des Präprozessors sowie der Prozeßablauf für ein Verfahren zur Integritätsprüfung und Sichtenaktualisierung während einer Transaktion veranschaulicht. Da es ein Ziel dieser Verfahren ist, SQL-basierte DB-Systeme funktional zu erweitern, werden die Verfahren mit den Implementierungen in DB-Systemen kommerzieller Hersteller verglichen. Abschließend erfolgt ein Vergleich mit anderen triggerbasierten Implementierungen und eine Diskussion der Eignung dieser Triggerausführungsmodelle für die Realisierung inkrementeller Propagierungsverfahren in SQL.

Kapitel 2

Relationale Datenbanksprachen

In diesem Kapitel werden Grundlagen deduktiver relationaler Datenbanken und DB-Sprachen zusammengestellt, soweit sie im Kontext der Sichtenanpassung und der Integritätsprüfung in Datenbanken relevant sind. Dort, wo Konzepte fehlen bzw. keine einheitlich anerkannten Notationen existieren, werden neue vorgeschlagen.

Datalog und SQL3 sind auf den ersten Blick recht unterschiedliche relationale Sprachen.

- Datalog ist eine kalkülreine Sprache, die auf dem Domänenkalkül (DRC, 'domain relational calculus') basiert. Mit nur den zwei logischen Operatoren Konjunktion und Negation entspricht Datalog einem syntaktisch vereinfachten DRC (Abschnitt 2.1).

  SQL ist eine Hybridsprache, die auf dem Tupelkalkül (TRC, 'tupel relational calculus') und der Relationalen Algebra (RA, 'relational algebra') basiert und sich durch eine Vielzahl semantisch redundanter Vergleichsoperatoren (z.B. EXISTS, IN, =ANY,...) und Teilausdrücke (z.B. <join_table_expr>, <nonjoin_table_expr>) auszeichnet. Die 'sprechenden' Befehlsbezeichner zielen darauf ab, intuitives Verständnis der Funktionalität der Anweisungen bei den Anwendern hervorzurufen (Abschnitt 2.2).

- Datalog ist eine nicht-standardisierte DB-Sprache. Lediglich für die Regelsprache (Abschnitt 2.1.1) hat sich ein Quasi-Standard in der Literatur durchgesetzt ([CGT90], [GrMi89], [Ull89], [Bid91]). Für eine Änderungssprache zur Manipulation von Fakten sind eine Vielzahl unterschiedlicher Konzepte diskutiert worden (Abschnitt 2.1.3). Eine Schemadefinitionssprache ist noch nicht vorgeschlagen worden.

  SQL liegt inzwischen in der dritten standardisierten Version (ANSI, ISO: [ANSI99I], [ANSI99II]) vor. Seit 1986, als der erste SQL-Standard verabschiedet wurde, ist die Anfrage-, Änderungs- und Schemadefinitionssprache permanent (SQL-86: 1986; SQL-92: 1992; SQL3: 1999) in ihrer Funktionalität weiterentwickelt und neuen Anforderungen angepaßt worden (z.B. aktive Regeln, objektrelationale DB-Objekte). Gleichzeitig hat sich SQL im kommerziellen Bereich als die am weitesten verbreitete DB-Sprache für RDBMS etabliert.

Bei näherer Betrachtung wird aber deutlich, wie grundlegend die Gemeinsamkeiten beider Sprachen sind:

- Datalog wie auch SQL sind deklarative Sprachen. Daß sie auf verschiedenen Kalkülen basieren, ist unproblematisch, da die Transformation von TRC-Ausdrücken in semantisch äquivalente DRC-Ausdrücke u.a. bereits Ullman in [Ull89] gezeigt hat.

- Datalog wie auch SQL sind relational vollständig (Abschnitte 2.1, 2.2). Da in SQL3 nun auch rekursive Sichten zulässig sind, ist analog zu den rekursiven Datalog-Regeln die transitive Hülle auch in SQL formulierbar.

- Analoge Konzepte werden in beiden Sprachen schnell offensichtlich, so z.B. Datalog-Regeln und SQL-Sichten, Datalog-Anfragen und SQL-SELECT-Ausdrücke (Abschnitt 3.2).

Da der Relationsbegriff im Rahmen dieser Arbeit von zentraler Bedeutung ist und dabei sehr vielschichtig verwendet wird, wird er vor der formalen Definition hinsichtlich einiger seiner verschiedenen Eigenschaften kurz diskutiert. Im relationalen Modell ist eine Relation eine Menge gleichstrukturierter Tupel (Fakten). Jedes Tupel einer Relation ist eindeutig identifizierbar. Eine Relation in SQL hingegen kann Duplikate enthalten und stellt somit eine Multimenge dar. Codd hat u.a. bereits in [Codd88] diese SQL-Eigenschaft als einen von drei 'fatalen Fehlern' in SQL bezeichnet, wobei er ausdrücklich betont, daß dies kein Fehler des relationalen Modells ist. Als problematisch sieht er in diesem Zusammenhang u.a. an, daß die Ergebnisse von Anfragen variieren können (Nichtdeterminismus) je nachdem, ob in den (Zwischen-)Ergebnissen Duplikate zugelassen sind oder nicht. Ich schließe mich der Argumentation von Codd an und diskutiere SQL unter der Annahme, daß eine Relation eine Menge von Tupeln ist (Abschnitt 2.2).

Bei der Klassifizierung der verschiedenen Arten von Relationen spielen vier Begriffspaare eine zentrale Rolle. Grundsätzlich sind diese Konzepte orthogonal zueinander anwendbar. Die Kombinationen bestimmter Konzepte sind jedoch wenig sinnvoll. Einige Konzepte hingegen werden durch andere impliziert.

- Basisrelation $\leftrightarrow$ abgeleitete Relation:
  Für eine **Basisrelation** (extensionale/EDB-Relation, SQL: Basistabelle[1]) wird in der Datenbank eine Relationsdefinition gespeichert. Dazu gehören u.a. die Namen der Relation und der Attribute (Spalten). Für eine **abgeleitete Relation** (intensionale/IDB-Relation, SQL: Sicht/View) wird zudem die Ableitungsvorschrift (Deduktionsregel, Qualifikationsteil einer Sichtdefinition) in der Datenbank gespeichert. Die Fakten einer abgeleiteten Relation werden anhand der Ableitungsvorschrift auf der Grundlage der jeweils vorliegenden Basisfakten ermittelt.

- virtuelle $\leftrightarrow$ materialisierte Sichten:
  Mittels Sichten werden auf der Grundlage der Basisfakten neue Fakten abgeleitet. Bei jeder Anfrage auf eine **virtuelle** Sicht werden die Fakten der Sicht immer wieder neu abgeleitet, was ggf. zur Performanzproblemen führen kann. Bei einer **materialisierten** Sicht werden die Fakten bei der Erstellung der Relation einmal ermittelt und dann gespeichert, so daß sie bei Anfragen nicht wiederholt abgeleitet werden müssen. Bei Änderungen der zugrunde liegenden Basisfakten müssen die materialisierten Sichtfakten aktualisiert werden.

[1] Der Begriff der Tabelle wird in SQL mit unterschiedlichen Bedeutungen verwendet. SQL-Anwender verwenden ihn im Sinne von Basistabellen, obwohl er vom Standard als Oberbegriff für Basistabellen und Sichten definiert ist (Abschnitt 2.2).

- persistente ↔ temporäre Relation
  Eine **persistente** Relation ist dauerhafter Bestandteil des DB-Schemas und muß explizit vom Entwickler gelöscht werden. **Temporäre** Relationen existieren nur vorübergehend während einer Modulausführung oder einer DB-Session und werden zum Ende der Modulausführung/Session automatisch vom DB-System gelöscht. Beide Konzepte sind grundsätzlich orthogonal zu den Konzepten der Basis-/abgeleiteten Relation und der virtuellen/materialisierten Sichten.

Neben den Relationen und Sichten sind Integritätsbedingungen im folgenden von zentraler Bedeutung. Integritätsbedingungen formulieren Konsistenzanforderungen an die Basisfakten und an die abgeleiteten Fakten, damit der zulässige DB-Zustand eine sinnvolle Situation der realen Welt repräsentiert. Die Einschränkung der zulässigen Werte von Attributen auf bestimmte Datentypen stellt eine Konsistenzanforderung dar, die als modellinhärente Integrität bezeichnet wird. Da die Implementierung von Typkonvergenzprüfungen zur Basisfunktionalität eines DB-Systems kommerzieller Hersteller gehört, wird sie im folgenden nicht weiter berücksichtigt.

Integritätsbedingungen können **prozedural** formuliert und dezentral in Anwendungsprogrammen geprüft werden. Die Überführung der vom Anwender vielfach deklarativ formulierten Anforderungen in Algorithmen ist für Entwickler vielfach nicht einfach und somit fehleranfällig. Zudem unterliegt auch die wiederholte Anwendung von Integritätsbedingungen in Anwendungsprogrammen den gleichen Problemen, wie sie ganz allgemein im Rahmen des Software Engineerings bei der Wiederverwendung von Software Komponenten auftreten. Diese Probleme motivieren **deklarativ** formulierte Integritätsbedingungen, die zentral in der Datenbank gespeichert und verwaltet werden. Die Integritätskomponente des DB-Systems gewährleistet die automatische Prüfung der Konsistenz, wenn Ereignisse (z.B. Einfügung, Löschung, Modifikation) auftreten, die die zu prüfende Faktenbasis manipulieren. Die Klassifizierung der Semantik von Integritätsbedingungen erfolgt im wesentlichen nach dem Kriterium der Anzahl der zu prüfenden DB-Zustände ([Das92], [Cre92], [HS95], [Lip88]).

- Mittels **statischer** Integritätsbedingungen werden Konsistenzanforderungen an einen DB-Zustand ausgedrückt.

- Anforderungen an die Konsistenz von Zustandsübergängen werden durch **dynamische** Integritätsbedingungen ausgedrückt. Zu diesem Zweck, muß die zugrunde liegende Sprache um eine geeignete temporale Syntax erweitert werden (Zugriff auf alte/neue Werte von Fakten, temporale Quantoren). Mittels **transitionalen** Integritätsbedingungen werden Anforderungen an die Konsistenz eines Zustandsübergangs ausgedrückt und mittels **temporalen** Integritätsbedingungen an eine Folge von Zustandsübergängen.

Die nachfolgende Gliederung dieses Kapitels ist für SQL-Anwender ungewohnt. In SQL sind Integritätsbedingungen und Sichten persistente Schemaobjekte und werden mittels DDL-Befehlen erzeugt. Anfragen stellen temporäre und unbenannte DB-Objekte dar. Die drei Konzepte werden in Abschnitt 2.1 für Datalog gemeinsam behandelt, weil sich in Datalog komplexere Integritätsbedingungen und Anfragen nur mit Hilfe von Regeln formulieren lassen. In SQL zählen die Anfragebefehle zur Datenmanipulationssprache (DML). Da aber Anfragen die Faktenmenge nicht beeinflussen, werden in Abschnitt 2.2

neben einer Änderungs- auch eine Anfragesprache definiert. Zur Erweiterung der beiden deduktiven DB-Sprachen werden zum Ende der beiden Abschnitte verschiedene Triggerkonzepte erörtert. Viele der nachfolgend definierten Begriffe werden sowohl im Rahmen von Datalog als auch von SQL verwendet. Die Bezeichner der Begriffe werden um den Namen der Sprache ergänzt, in dessen Kontext sie jeweils definiert wurden. Fehlt dieser Zusatz, so geht aus dem Gesamtzusammenhang hervor, welche Definition gemeint ist.

2.1 Datalog

Grant und Minker datieren in [GrMi89] die Anfänge der wissenschaftlichen Diskussion einer Verbindung von künstlicher Intelligenz und Datenbanken als eigenständiges Forschungsgebiet und die Entwicklung von Datalog auf die Mitte der 70er Jahre. Als grundlegende Arbeiten, die Syntax und Semantik von Datalog bestimmt haben, gelten u.a. [vEK76], [GaMi78]. Bislang hat sich nur bzgl. der Regelsprache (Abschnitt 2.1.1), ein Quasi-Standard etabliert, deren Semantik in Abschnitt 2.1.2 definiert wird. Für die beiden anderen Teilsprachen Datenmanipulation (DML, Abschnitt 2.1.3) und Datendefinition wie auch für eine Erweiterung um Trigger (Abschnitt 2.1.4) zeichnen sich noch keine allgemein anerkannten Konzepte ab, so daß eine Datalog-DDL und -DML definiert werden, wie sie für eine Diskussion von Propagierungsverfahren in Datalog angemessen sind. Als Triggerkomponente wird ActLog vorgestellt, da Griefahn in [Grie97] eine Implementierung für ein Datalog-basiertes Propagierungsverfahren entwickelt hat. Im weiteren ist mit der Bezeichnung Datalog immer Datalog mit Negation (Datalog$^\neg$) gemeint.

2.1.1 Regeln und Anfragen

Die Regel- und Anfragesprache wird hier in einer sehr grundlegenden Form mit rekursiven Regeln, Vergleichs-Built-ins und Negation aber ohne Erweiterungen wie z.B. Funktionen, Aggregate, Duplikate spezifiziert. Die Problematik der Erweiterung von Datalog um Funktionsaufrufe ist u.a. diskutiert worden in [AH88], [Ull89], [CGH94]. Sie ist aber nicht Gegenstand dieser Arbeit, bei der es in erster Linie um die grundsätzliche Vergleichbarkeit zwischen SQL und Datalog und um eine grundsätzliche Anwendbarkeit inkrementeller Verfahren im SQL-Kontext geht.

Eine Datalog-Bereichsvariable ist ein Großbuchstabe wie $X, X_1, ..., Y, Y_1, ..., Z, Z_1, ...$ Ein Datalog-Term t_i ist eine Konstante $a, b, c, \ldots, 1, 2, 3, \ldots$ oder eine Datalog-Bereichsvariable. Ein variablenfreier Term (Konstante) wird auch als Grundterm bezeichnet.

Definition 2.1.1 (atomare Datalog-Formel)
Eine atomare Datalog-Formel (Datalog-Atom) ist ein Ausdruck der Form

1. $p(t_1, \ldots, t_n)$,
   wobei p ein n-stelliges Relationssymbol ist mit $n \geq 0$ und $t_1, \ldots, t_n$ Terme sind,
   oder

2. $t_i \ \Theta \ t_j$,
   wobei t_i, t_j Terme sind und Θ ein Vergleichsoperator ($\Theta \in \{=, >, <, \geq, \leq, \neq\}$) ist.

Gemäß Punkt 1 definierte atomare Formeln werden als DB-Atome bezeichnet und gemäß Punkt 2 definierte atomare Formeln werden als Vergleichsatome bezeichnet. Da Datalog auf dem Domänenkalkül basiert, wird eine positionelle Schreibweise der Attribute einer Relation zugrunde gelegt. Die Attribute von Datalog-Relationen werden nicht benannt.

Ist A ein Datalog-Atom, so spezifiziert $rel(A)$ das Relationssymbol von A und $vars(A)$ die Menge der nicht grundinstanziierten Datalog-Terme (Bereichsvariablen) von A. Ist A frei von Variablen ($vars(A) = \emptyset$), so wird A als Grundatom (Fakt) bezeichnet.

Definition 2.1.2 (Datalog-Formel)
Eine Datalog-Formel F ist

1. *eine aussagenlogische Konstante* TRUE *oder* FALSE*, oder*
2. *eine positive oder negative atomare Datalog-Formel (Literal) L bzw. $\neg L$, oder*
3. *eine Konjunktion von Literalen $L_1 \wedge ... \wedge L_n$,*
   wobei die Literale L_i für $1 \leq i \leq n$ positive oder negative atomare Datalog-Formeln sein können[2]*.*

Ist F eine Datalog-Formel, so spezifiziert $rel(F)$ die Menge der Relationssymbole, die in den Literalen von F auftreten, und $vars(F)$ die Menge der Variablen, die in F auftreten. Ist F frei von Variablen ($vars(F) = \emptyset$), so wird F als grundinstanziiert bezeichnet.

Anders als in Datalog üblich kann in Anlehnung an SQL eine Ableitungsregel optional über eine WITH-Klausel mit Hilfsregeln verfügen.

Definition 2.1.3 (Ableitungsregel)
Eine Ableitungsregel (Deduktionsregel, passive Regel) ist ein Ausdruck der Form

$$H \leftarrow B \quad [\ \textit{WITH}\ \ R_1^{help} \leftarrow B_1\ ,\ldots,\ R_m^{help} \leftarrow B_m\ ];$$

mit einem DB-Atom H als Kopfliteral und einer Datalog-Formel B als Regelrumpf. Die WITH*-Liste mit den Hilfsrelationen R_i^{help} ist optional ($0 \leq i \leq m$). $R_i^{help} \leftarrow B_i$ ist eine Hilfsregel mit dem DB-Atom R_i^{help} als Kopfliteral und einer Datalog-Formel B_i als Regelrumpf.*

Die Hilfsregeln R_i^{help} sind keine eigenständigen DB-Objekte, sondern besitzen nur einen lokal und temporär begrenzten Gültigkeitsbereich. Als Rumpfliterale können die Kopfliterale dieser Hilfsregeln R_i^{help} nur im Regelrumpf von B oder im Rumpf einer anderen Hilfsregel der gleichen Regelspezifikation verwendet werden. Jede dieser Hilfsregelspezifikationen unterliegt den gleichen syntaktischen Einschränkungen wie eine Regelspezifikation mit der Ausnahme, daß sie über keine eigene WITH-Klausel verfügen können.

[2]Damit die Analogie zur Konjunktion in RA, TRC, DRC und SQL anschaulich wird, wird hier die Konjunktion von Rumpfliteralen explizit mit dem logischen $\wedge$-Junktor ausgedrückt. In vielen anderen Publikationen wird diese Konjunktion durch ein Komma ',' spezifiziert.

Ein Fakt ist ein grundinstanziiertes DB-Atom $h(t_1, \ldots, t_n)$, wobei h der Name der Relation ist und die Terme $t_1, \ldots, t_n$ Konstanten sind. Fakten lassen sich auch als eine spezielle Regel der Form $h(t_1, \ldots, t_n) \leftarrow$ TRUE darstellen. Variablen in einer Datalog-Regel, die nicht im Regelkopf auftreten, gelten als zu Beginn des Regelrumpfs implizit existenzquantifiziert. Die Variablen, die im Kopfliteral H einer Regel $H \leftarrow B$ auftreten, sind die freien Variablen der Regel ($free(H \leftarrow B) := vars(H)$). Die Variablen, die ausschließlich im Regelrumpf auftreten, werden als gebundene Variablen bezeichnet ($bound(H \leftarrow B) := vars(B) \setminus vars(H)$).

Eine wichtige Voraussetzung dafür, daß eine Regel über eine sinnvolle Semantik verfügt bzw. in effizienter Weise ausgewertet werden kann, ist die Forderung der Bereichsbeschränkung ('range-restricted', 'safe', sicher) ([Ull89]).

Definition 2.1.4 (beschränkte Variable)
In einer Regel $H \leftarrow L_1 \wedge \ldots \wedge L_n$ tritt eine Variable $X_i \in vars(H) \cup (\bigcup_{i=1}^{n} vars(L_i))$ beschränkt (limited) auf, gdw.

1. *X_i in mindestens einem positiven Rumpfliteral L_i auftritt, daß ein DB-Atom ist, oder*

2. *X_i in mindestens einem positiven Rumpfliteral L_i auftritt, daß ein Vergleichs-Built-in der Form $X_i = Y$ ist, und Y entweder eine Konstante ist oder selbst eine beschränkte Variable oder*

3. *X_i in mindestens einem negativen Rumpfliteral L_i auftritt, daß ein Vergleichs-Built-in der Form $X_i \neq Y$ ist, und Y entweder eine Konstante ist oder selbst eine beschränkte Variable.*

Definition 2.1.5 (bereichsbeschränkte Regel)
Eine Regel $H \leftarrow L_1 \wedge \ldots \wedge L_n$ ist bereichsbeschränkt (sicher, 'allowed', 'safe'), gdw. jede Variable

$$X_i \in vars(H) \cup (\bigcup_{i=1}^{n} vars(L_i))$$

beschränkt ist.
Eine Regel mit Hilfsregeln

$$H \leftarrow L_1 \wedge \ldots \wedge L_n \ \textit{WITH}\ R_i^{help} \leftarrow B_i \quad \textit{mit}\ 1 \leq i \leq m$$

ist bereichsbeschränkt, gdw. die Regel H und jede Hilfsregel R_i^{help} mit $1 \leq i \leq m$ bereichsbeschränkt sind.

Die syntaktische Besonderheit einer normalisierten Schreibweise der Existenzquantifizierung in Verbindung mit Negation ist einer der Gründe für die Problematik der Bereichsbeschränkung in Datalog. Mit folgender Konsequenz ist Negation nur unmittelbar vor Rumpfliteralen zulässig und nicht vor impliziten Quantoren.

Beispiel 2.1.6 (Implizite Quantifizierung in Datalog-Regeln)
Die folgende Regel h ist aufgrund der fehlenden positiven Bindung der Z-Variablen nicht bereichsbeschränkt:

$$h(X) \leftarrow l_1(X,Y) \wedge \neg\, l_2(X,Y,Z).$$

Aufgrund der impliziten Existenzquantifizierung ist die Regel wie folgt definiert:

$$h(X) \leftarrow \exists Y,Z \quad l_1(X,Y) \wedge \neg\, l_2(X,Y,Z).$$

Intuitiv gemeint ist aber folgende Datalog-Regel:

$$h(X) \leftarrow \exists Y \quad l_1(X,Y) \wedge \neg\exists Z \quad l_2(X,Y,Z).$$

Als bereichsbeschränkte Regel läßt sich diese Regel nur mit einer Hilfsrelation definieren, die die Tupel der Relation l_2 auf die gebundenen Variablen X,Y beschränkt:

$$h(X) \leftarrow l_1(X,Y) \wedge \neg\, h_1(X,Y); \qquad h_1(X,Y) \leftarrow l_2(X,Y,Z);\ .$$

Damit in solchen Fällen die Regelmenge nicht unnötig vergrößert und schwerer zu warten wird, bietet es sich an, solche Hilfsregeln innerhalb der WITH*-Option von h zu definieren:*

$$h(X) \leftarrow l_1(X_1,X_2) \wedge \neg\, h_1(X,Y) \quad \textit{WITH} \quad h_1(X,Y) \leftarrow l_2(X,Y,Z).$$

Die syntaktisch simplifizierte, auf dem Bereichskalkül basierende Datalog-Syntax ohne Disjunktionsoperator hat zur Folge, daß Datalog zwar relational vollständig, aber nicht streng relational vollständig[3] ist. Ullman zeigt in [Ull89], daß sichere, hierarchische Datalog-Regeln mit Negation relational vollständig sind ([CGT90]). Daß sicheres Datalog anders als sicheres DRC oder sicheres TRC streng relational vollständig ist, resultiert aus der Besonderheit, daß disjunktive Verknüpfungen auf der Basis der Vereinigungssemantik von Regeln ausgedrückt werden.

In Datalog können, anders als in SQL, **parameterlose Relationen** definiert werden. Das zugehörige Relationssymbol verfügt über keine Argumente (Definition 2.1.1 atomare Datalog-Formel ohne Terme ($n = 0$)). Das Konzept parameterloser, abgeleiteter Relationen ist insofern wichtig, als mit ihrer Hilfe der Wahrheitswert von Bedingungen getestet werden kann. Ist die Bedingung im Rumpf einer parameterlosen Regel wahr, so wird das Ergebnis der Anwendung dieser Regel als TRUE interpretiert anderenfalls als FALSE.

Parameterlose Regeln mit nur einem Rumpfliteral können als **Integritätsbedingung** deklariert werden. Für komplexere Formeln kann dieses Rumpfliteral das Kopfliteral einer Hilfsregel der Integritätsbedingung sein. Motiviert wird diese ausschließlich syntaktische Einschränkung durch eine vereinfachten Abbildung nach SQL und einer vereinfachten Relevanzanalyse im Rahmen inkrementeller Verfahren. Die Integritätskomponente des Datalog-DB-Systems wendet die als Integritätsbedingungen spezifizierten Regeln zum Transaktionsende an, interpretiert die Ergebnisse und reagiert entsprechend. Ist das Ergebnis für mindestens eine Integritätsbedingung FALSE, so wird die Transaktion zurückgerollt, anderenfalls werden die Änderungen gespeichert.

[3] Eine DB-Sprache ist relational vollständig, gdw. jeder Term der RA in der anderen Sprache formuliert werden kann. Sie ist streng relational vollständig, gdw. jeder Term der RA durch einen einzigen Ausdruck in der anderen Sprache formuliert werden kann [Cre92].

Beispiel 2.1.7 (Datalog-Integritätsbedingungen)

- *Allaussage: Inklusionsbeziehung zwischen den Relationen p, q!*

  Prädikatenlogik: $\forall X \mid p(X) \Rightarrow q(X) \quad \Leftrightarrow \quad \neg\exists X \mid \neg(p(X) \Rightarrow q(X)) \quad \Leftrightarrow \quad \neg\exists X \mid p(X) \wedge \neg q(X)$

  Datalog: *CONSTRAINT* $ic_1 \leftarrow \neg h_1$ *WITH* $h_1 \leftarrow p(X) \wedge \neg q(X)$; .

- *Existenzaussage: Es gibt ein p-Fakt mit dem Wert 5 an der 2. Argumentposition!*

  Prädikatenlogik: $\exists X \mid p(X, 5)$

  Datalog: *CONSTRAINT* $ic_2 \leftarrow p(X, 5)$; .

Diese Notation einer Datalog-Integritätsbedingung basierend auf der impliziten Existenzquantifizierung ist aufgrund ihrer Analogie zur Syntax einer SQL-ASSERTION gewählt worden, bei der eine Allaussage ebenfalls nur mittels eines NOT EXISTS-Operator formuliert werden kann. Eine andere vielfach verwendete regelbasierte Darstellungsform ist die sogenannte 'Denial'-Form ([TO95], [Oli91], [Das92], [AIP96]). Sie setzt die implizite Allquantifizierung der freien Variablen im Kopfliteral voraus und stellt somit eine Abweichung von der konstruktiven Semantik einer Regel mit freien Variablen dar.

Datalog wird in der Literatur häufig nur als Regelsprache spezifiziert. Wenn Anfragen spezifiziert werden, so werden im allgemeinen unterschiedliche Notationen verwendet. Vielfach basieren sie auf regelorientierten Darstellung. Einige Autoren verwenden unbenannte Regeln ($\leftarrow \; L_1 \wedge, \ldots \wedge \; L_n$), um sowohl den Deduktionscharakter als auch die mangelnde Persistenz des Anfrageergebnisses zu unterstreichen ([GrMi89], [Grie97], [CGH94], [Das92], [AIP96]). Andere Autoren ([CGT90], [BK98]) verwenden in Anlehnung an Prolog als Symbol für eine Anfrage das Fragezeichen ($? - \; L_1 \wedge, \ldots \wedge \; L_n$). Die im folgenden verwendete Syntax unterstreicht den Mengencharakter einer Anfrage. Eine Anfrage, deren Bedingung B für alle Fakten zu FALSE ausgewertet wird, liefert die leere Menge als Ergebnis. Analog zur Darstellung der parameterlosen Regeln, wird für eine Existenzanfrage eine parameterlose Zielliste ($n = 0$) spezifiziert. Ist eine Existenzanfrage erfüllt, so enthält die Ergebnismenge das leere Tupel. Dieses Ergebnis wird als TRUE und die leere Menge als FALSE interpretiert. Die bei den Ableitungsregeln (Definition 2.1.3) vorgestellte WITH-Option mit lokalen Hilfsregeln wird für Anfragen analog definiert.

Definition 2.1.8 (Bereichsbeschränkte Datalog-Anfrage)
Eine bereichsbeschränkte Datalog-Anfrage ist ein Ausdruck der Form:

$$\{ \; (t_1, \ldots, t_n) \;\; | \;\; B \; \} \;\; [\;\; \textit{WITH} \; H_1^{help} \leftarrow B_1 \; , \ldots , \; H_m^{help} \leftarrow B_m \;],$$

wobei die Terme $t_1, \ldots, t_n$ $(n \geq 0)$ Konstanten oder freie Variablen der Formel B sind $(t_i \in free(B), \; 1 \leq i \leq n)$ und B eine Datalog-Formel ist. Alle Variablen $X \in vars(t_1, \ldots, t_n) \cup vars(B)$ sind bereichsbeschränkt. Die WITH-Liste mit den bereichsbeschränkten Hilfsregeln $H_j^{help} \leftarrow B_j$ $(1 \leq j \leq m)$ ist optional.

Im allgemeinen wird keine Datalog-DDL spezifiziert sondern nur entsprechende Annahmen formuliert. Damit aber eine Analogie zu SQL gewahrt wird, werden DDL-Anweisungen für die drei im Rahmen dieser Arbeit wichtigsten DB-Objekte definiert. In Ermangelung von Attributsymbolen werden die Attribute einer Relation über ihre Parameterpositionen identifiziert (positionelle Darstellung). Sei q eine Basisrelation mit n Attributen. R sei eine bereichsbeschränkte Datalog-Regel mit dem Relationssymbol p des Kopfliterals und IC sei eine parameterlose Datalog-Regel mit dem Kopfliteral ic und mit genau einem ebenfalls parameterlosen Rumpfliteral. Werden die nachfolgenden DDL-Anweisungen ausgeführt, dann werden

- eine Basisrelation q mit n Attributen: CREATE BASE RELATION q : n;
- eine virtuelle Sicht p erzeugt: CREATE VIRTUAL VIEW R,
- eine materialisierte Sicht p erzeugt: CREATE MATERIALIZED VIEW R,
- eine Integritätsbedingung IC erzeugt: CREATE CONSTRAINT ic.

Definition 2.1.9 (Datalog-Datenbankschema)
Ein Datalog-Datenbankschema ist ein Tripel $\mathcal{S}^{\mathcal{D}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C} \rangle$ *mit*

1. *einer endlichen, ggf. leeren Menge von Basisrelationen* $\mathcal{B}$,
2. *einer endlichen, ggf. leeren Menge stratifizierbarer Dedukttionsregeln* $\mathcal{R} := \mathcal{RM} \cup \mathcal{RV} \cup \mathcal{RH}$,
   - $\mathcal{RM}$ *ist die Menge der Regeln für materialisierte Sichten,*
   - $\mathcal{RV}$ *ist die Menge der Regeln für virtuelle Sichten,*
   - $\mathcal{RH}$ *ist die Menge der Hilfsregeln* R_i^{help} *aller Regeln* $R_j \in \mathcal{RM} \cup \mathcal{RV}$
3. *und einer endlichen, ggf. leeren Menge* $\mathcal{C}$ *von Integritätsbedingungen.*

Definition 2.1.10 (Datalog-Signatur)
Sei $\mathcal{S}^{\mathcal{D}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C} \rangle$ *ein Datalog-Datenbankschema, dann ist die Signatur des Schemas ein Tupel* $\Sigma^D = \langle PS, AS \rangle$ *mit*

1. *einer endlichen, ggf. leeren Menge von Prädikatsymbolen* $\mathcal{PS}$, *wobei* $\mathcal{PS} := rel(\mathcal{B}) \cup rel_{virt}(\mathcal{R}) \cup rel_{mat}(\mathcal{R}) \cup rel_{help}(\mathcal{R})$
2. *und einer endlichen, ggf. leeren Menge* $\mathcal{AS}$ *mit Tupeln der Form:* $\langle p, n \rangle$, *wobei* $p \in rel(R)$ *und* n *die Anzahl der Parameter von* p *ist.*

Definition 2.1.11 (Deduktive Datalog-Datenbank)
Eine deduktive Datalog-Datenbank ist ein Tupel $\mathcal{D}^{\mathcal{D}} = \langle \mathcal{F}, \mathcal{S}^{\mathcal{D}} \rangle$ *mit*

1. *einer endlichen, ggf. leeren Menge* $\mathcal{F}$ *von Grundatomen der Basisrelationen (Basisfakten, EDB-Fakten)*
2. *und dem Datenbankschema* $\mathcal{S}^{\mathcal{D}}$.

$rel_{body}(R)$ ist die Menge der Relationssymbole der Rumpfliterale einer Regel $R \in \mathcal{R} \cup \mathcal{C}$ einschließlich aller Rumpfliterale der Hilfsregeln von R und $rel_{head}(R)$ das Relationssymbol des Kopfliterals ($rel(R) := rel_{head}(R) \cup rel_{body}(R)$). $rel_{edb}(\mathcal{R})$ ist die Menge der Relationssymbole aller Basisrelationen, die in den Regelrümpfen einer Regelmenge $\mathcal{R}$ auftreten. $rel_{idb}(\mathcal{R})$ ist die Menge aller Relationssymbole abgeleiteter Relationen einer Regelmenge $\mathcal{R}$, die sich aus den drei disjunkten Teilmengen der Relationssymbole materialisierter $rel_{mat}(\mathcal{R})$ und virtueller Sichten $rel_{virt}(\mathcal{R})$ und der Menge der Relationssymbole der Hilfsregeln $rel_{help}(\mathcal{R})$ zusammensetzt. Die Relationssymbole der verschiedenen Relationsarten sind paarweise disjunkt.

$$rel_{idb}(\mathcal{R}) := rel_{virt}(\mathcal{R}) \cup rel_{mat}(\mathcal{R}) \cup rel_{help}(\mathcal{R}).$$

Definition 2.1.12 (Abhängigkeitsgraph für Relationen)
Sei $\mathcal{D}^{\mathcal{P}}$ eine deduktive Datenbank mit der Menge der Relationssymbole $\mathcal{PS}$ und $\mathcal{E}$ sei die Menge der gerichteten und beschrifteten Kanten ($\mathcal{E} := \mathcal{PS} \times \mathcal{PS} \times \{+, -\}$), dann ist der Abhängigkeitsgraph ein gerichteter Graph $G_D := (\mathcal{PS}, \mathcal{E})$.
Sei eine (Hilfs-)Regel $H \leftarrow L_1 \wedge \ldots \wedge L_n$ gegeben mit den Relationssymbolen $p = rel(H)$ und $q = rel(L_i)$ $(1 \leq i \leq n)$, dann enthält G_D eine gerichtete Kante von q nach p mit der Beschriftung:

- *$+$, wenn L_i ein positives Literal ist:* $(q,\ p,\ +)$,
- *$-$, wenn L_i ein negatives Literal ist:* $(q,\ p,\ -)$.

Als Basis für die Prüfung der Relevanz von Regeln und Integritätsbedingungen für die Modifikation eines Attributwerts, kann ein Abhängigkeitsgraph für die Attribute von Relationen definiert werden. Der Begriff Abhängigkeit wird in diesem Zusammenhang mit einer erweiterten Bedeutung verwendet. Ein Attribut eines Kopfliterals ist abhängig von einem Attribut eines Rumpfliterals, wenn Wertänderungen des Rumpfattributs auch Wertänderungen des Kopfattributs zur Folge haben. So gilt z.B. für die Regel $p(X) \leftarrow q(X, Y, Z) \wedge Z > 13$, daß die Variable X gemäß Punkt 1 und Z gemäß Punkt 2 abhängig ist. Für Y gilt keine der Abhängigkeiten.

Definition 2.1.13 (Abhängigkeitsgraph für Attribute)
Sei $\mathcal{D}^{\mathcal{P}}$ eine deduktive DB mit der Menge der Relationssymbole $\mathcal{PS}$ und der Menge der Variablensymbole $\mathcal{V} := vars(\mathcal{R})$. $\mathcal{N}$ sei die Menge der natürlichen Zahlen, die die Argumentpositionen i_p, i_q von Variablen bezeichnet, dann ist der Attributabhängigkeitsgraph ein gerichteter Graph

$$\mathcal{E} := \mathcal{PS} \times \mathcal{V} \times N \times \mathcal{PS} \times \mathcal{V} \times \mathcal{N} \times \{+, -\}.$$

Für jedes Tripel $(q,\ p,\ +)$ bzw. $(q,\ p,\ -)$ aus einem Abhängigkeitsgraphen G_D enthält der Attributabhängigkeitsgraph G_D^{Attr} mindestens eine gerichtete Kante von q nach p, wenn[4]

1. *beide Literale gemeinsame Variablen aufweisen ($X \in vars(p) \cap vars(q)$):*

$$(q,\ X,\ i_q,\ p,\ X,\ i_p,\ +) \qquad \textit{bzw.} \qquad (q,\ X,\ i_q,\ p,\ X,\ i_p,\ -),$$

[4]Die Werte der mit dem Unterstrich ('_') markierten Parameterpositionen sind nicht relevant.

2. *die Variable X im Rumpfliteral q gebunden ist und zudem in weiteren Rumpfliteralen der Regel auftritt:*

$$(q,\ X,\ i_q,\ p,\ \_,\ \_,\ +) \qquad \textit{bzw.} \qquad (q,\ X,\ i_q,\ p,\ \_,\ \_,\ -),$$

3. *die Variable X Operand eines Vergleichsatoms ist und entweder im Kopfliteral p oder im Rumpfliteral q gebunden ist ($X \in vars(p) \cup vars(q)$):*

$$\begin{array}{llll} (q,\ X,\ i_q,\ \_,\ \_,\ \_,\ +) & \textit{bzw.} & (q,\ X,\ i_q,\ \_,\ \_,\ \_,\ -) & \textit{oder} \\ (\_,\ \_,\ \_,\ p,\ X,\ i_p,\ +) & \textit{bzw.} & (\_,\ \_,\ \_,\ p,\ X,\ i_p,\ -). & \end{array}$$

Aufgrund der optionalen WITH-Klausel müssen die im Datalog-Kontext üblichen Definitionen von Abhängigkeiten, die keine Hilfssichten berücksichtigen, angepaßt werden. Da die im weiteren vorgestellten inkrementellen Verfahren auf der Bearbeitung persistenter DB-Objekte basieren und auch dort das Konzept der Hilfssichten noch nicht behandelt wurde, kommt dem Begriff der unmittelbaren Abhängigkeit zwischen zwei persistenten Sichten eine besondere Bedeutung für zu. Die Abhängigkeiten zwischen den Argumentpositionen zweier Relationen definieren sich auf der Basis der Kanten des Attributabhängigkeitsgraphen G_D^{Attr} analog zu den Abhängigkeiten zwischen Relationen.

Definition 2.1.14 (Abhängigkeiten)
Sei $\mathcal{S}^{\mathcal{D}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C} \rangle$ ein DB-Schema von $\mathcal{D}^{\mathcal{D}}$ mit der Menge der Hilfssichten $\mathcal{RH}$ und einem Abhängigkeitsgraphen G_D. $p \in rel(\mathcal{R} \cup C)$ sei das Symbol einer Integritätsbedingung oder Sicht und $q \in rel(\mathcal{R})$ sei ein Relationssymbol,

1. *dann hängt p von q ab ($p < q$), gdw.*
   der Abhängigkeitsgraph G_D einen Pfad beliebiger Länge von q nach p enthält.

2. *dann hängen p und q gegenseitig voneinander ab ($p \approx q$), gdw.*
   $p < q$ und $q < p$.

3. *dann hängt p von q positiv ($p <_+ q$) / negativ ($p <_- q$) ab , gdw.*
   $p < q$ und alle Pfade in G_D eine gerade / ungerade Anzahl negativer Kanten enthalten.

4. *dann hängt p von q unmittelbar ab ($p <_1 q$), gdw.*
   in G_D eine Kante von q nach p der Art $(q, p, \star) \in G_D$ mit $\star \in \{+, -\}$ existiert.

5. *und sind $p, q \notin (\mathcal{RH})$, dann hängt p unmittelbar von q ab ($p <_1 q$), gdw.*
   $p < q$ und alle Knoten auf allen Pfaden von q nach p in G_D Hilfsrelationen repräsentieren.

6. *dann hängt p von q unmittelbar positiv ($p <_{+1} q$) / negativ ($p <_{-1} q$) ab, gdw.*
   in G_D eine Kante von q nach p der Art $(q, p, +)$ / $(q, p, +)$ existiert.

7. *und sind $p, q \notin (\mathcal{RH})$, dann hängt p von q unmittelbar positiv ($p <_+ q$) / negativ ($p <_- q$) ab, gdw.*
   $p < q$ und alle Pfade in G_D eine gerade / ungerade Anzahl negativer Kanten enthalten und alle Knoten auf allen Pfaden von q nach p in G_D Hilfsrelationen repräsentieren.

Damit bei der Ableitung der durch eine Regelmenge $\mathcal{R}$ beschriebenen abgeleiteten Fakten Rekursion und Negation sinnvoll gehandhabt werden können, ist es wichtig, für $\mathcal{R}$ eine Stratifikation zu erzeugen. Sie ist ebenfalls essentiell hinsichtlich der Propagierung von Folgeänderungen.

Definition 2.1.15 (Stratifikation)
Sei $\mathcal{R}$ eine Menge deduktiver Regeln (incl. lokaler Hilfsregeln R_i^{help}). Eine Stratifikation λ bzgl. $\mathcal{R}$ ist eine Abbildung der Menge der Relationssymbole $rel(\mathcal{R})$ auf eine Menge von positiven natürlichen Zahlen $\mathcal{N}$. Für alle Relationssymbole $p, q \in rel(\mathcal{R})$ gilt:

1. $p \in rel_{edb}(\mathcal{R}) \iff \lambda(p) = 0$
2. $p \in rel_{idb}(\mathcal{R}) \iff \lambda(p) \geq 1$
3. $p < q \ \wedge \ \neg(q < p) \iff \lambda(p) > \lambda(q)$
4. $p \approx q \iff \lambda(p) = \lambda(q)$.

Definition 2.1.16 (Klassen von Regelmengen, Datenbanken)
Sei $\mathcal{D}^{\mathcal{D}}$ eine deduktive DB mit der Menge deduktiver Regeln $\mathcal{R}$.

1. *Die Regelmenge/DB heißt positiv, gdw. keine Relationssymbole $p, q \in rel(\mathcal{R})$ existieren, die negativ von einander abhängen (kein: $p <_- q$).*
2. *Die Regelmenge/DB heißt semi-positiv, gdw. negative Abhängigkeiten nur von Basisrelationen existieren ($p <_- q \Rightarrow q \in rel_{edb}(\mathcal{R})$)*
3. *Die Regelmenge/DB heißt hierarchisch (nicht rekursiv), gdw. kein Relationssymbol $p \in rel(\mathcal{R})$ existiert, das von sich selbst abhängt (kein: $p \approx p$).*
4. *Die Regelmenge/DB heißt rekursiv, gdw. mindestens ein Relationssymbol $p \in rel(\mathcal{R})$ existiert, das von sich selbst abhängt ($p \approx p$).*
5. *Die Regelmenge/DB $\mathcal{R}$ heißt stratifizierbar, gdw. kein Relationssymbol $p \in rel(\mathcal{R})$ existiert, das von sich selbst negativ abhängt (kein: $p <_- p$).*

Das nachfolgende Beispiel eines Schemas einer Unternehmensstruktur wird als wiederkehrendes Beispiel die zentralen Themen dieser Dissertation veranschaulichen.

Beispiel 2.1.17 (Unternehmensorganisation)
Um das zu dem Schema aus Abbildung 2.1 gehörende DB-Schema $\mathcal{S}^{\mathcal{D}}$ zu erzeugen, müssen die folgenden Anweisungen in einer Datalog-Datenbank ausgeführt werden. Die Basisrelation 'mitarbeiter' verfügt über 4 Attribute (Name, Abteilungsnummer, Gehalt, Managername) und 'abteilung' über 2 (Abteilungsnummer, Bezeichnung):

CREATE BASERELATION mitarbeiter : 4;
CREATE BASERELATION abteilung : 2;

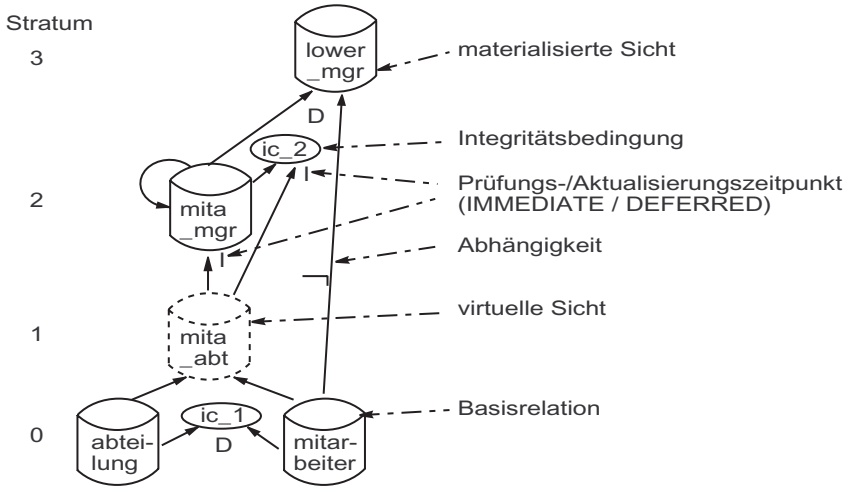

Abbildung 2.1: Mitarbeiter-Schema eines Unternehmens

Zur Erzeugung der Sichten werden folgende DDL-Anweisungen ausgeführt:

- *virtuelle Sicht 'mita_abt': Mitarbeiter-Informationen und das Abteilungskürzel:*

  CREATE VIRTUAL VIEW
  $mita\_abt(X_1, X_2, Y_2, X_4) \leftarrow mitarbeiter(X_1, X_2, X_3, X_4) \wedge abteil(X_2, Y_2);$

- *materialisierte Sicht 'mita_mgr': Mitarbeiter mit ihren (in-)direkten Managern:*

  CREATE MATERIALIZED VIEW
  $mita\_mgr(X_1, X_3, X_4) \leftarrow mita\_abt(X_1, X_2, X_3, X_4);$
  $mita\_mgr(X_1, X_2, Y_4) \leftarrow mita\_mgr(X_1, X_2, X_3) \wedge mita\_abt(X_3, Y_2, Y_3, Y_4);$

- *materialisierte Sicht 'lower_mgr': Mitarbeiter und ihre Manager, die dem unteren Management angehören (nicht der Abteilung 'Management' (Nr. 13))*

  CREATE MATERIALIZED VIEW
  $lower\_mgr(X_1, X_2, X_3) \leftarrow mita\_mgr(X_1, X_2, X_3) \wedge \neg h_1(X_3)$
  WITH $h_1(Y_1) \leftarrow mitarbeiter(Y_1, 13, Y_3, Y_4);$

- *Integritätsbedingung ic_1: Inklusionsbedingung des Attributs 'mitarbeiter.abteil' zu 'abteilung.nr':*

  CREATE CONSTRAINT
  $ic_1 \leftarrow \neg h_1$ *WITH* $h_1 \leftarrow mitarbeiter(X_1, X_2, X_3, X_4) \wedge \neg h_2(X_2);$
  $h_2(Y_1) \leftarrow abteilung(Y_1, Y_2);$

- *Integritätsbedingung ic_2: Manager müssen der gleichen Abteilung wie die Mitarbeiter angehören oder dem 'Management' ('MG').*

  CREATE CONSTRAINT
  $ic_2 \leftarrow \neg h_1$ *WITH* $h_1 \leftarrow mita\_mgr(X_1, X_2, X_3) \wedge \neg h_2(X_3, X_2);$
  $h_1 \leftarrow mita\_mgr(X_1, X_2, X_3) \wedge \neg h_2(X_3, 'MG');$
  $h_2(Y_1, Y_2) \leftarrow mita\_abt(Y_1, Y_2, Y_3);$

2.1.2 Semantik deduktiver Datenbanken

Die Semantik einer relationalen Datenbank ist vollständig durch die in der Datenbank gespeicherten Basisfakten $\mathcal{F}$ definiert, die Semantik einer deduktiven Datenbank durch die Basisfakten $\mathcal{F}$ und den gegebenen Deduktionsregeln $\mathcal{R}$. Für die Bestimmung der Menge der gespeicherten und ableitbaren Fakten $\mathcal{F}^\star$ einer Datenbank wird eine konstruktive Fixpunktsemantik vorgestellt ([vEK76], [Bid91], [Grie97]). Die nachfolgenden Definitionen der Auswertung von Regeln/Anfragen sind nicht nur hinsichtlich einer konstruktiven Fixpunktsemantik von zentraler Bedeutung sondern auch für die Semantik von Änderungsanweisungen (Abschnitt 2.1.3). Im weiteren wird mit $\mathcal{D}^{\mathcal{D}} = \langle \mathcal{F}, \mathcal{S}^{\mathcal{D}} \rangle$ eine deduktive Datalog-DB bezeichnet, wobei das DB-Schema definiert ist als $\mathcal{S}^{\mathcal{D}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C} \rangle$.

Eine Variablensubstitution (X/t) einer Variablen X ist die Ersetzung dieser Variablen durch einen Term t. Die Substitution $\sigma := \{v_1/t_1, \ldots, v_n/t_n\}$ von Variablen eines Ausdrucks A (Atom, Regel, Anfrage) mit $vars(A) = \{v_1, \ldots, v_n\}$ ist eine Menge von Variablensubstitutionen v_i/t_i mit $1 \leq i \leq n$. Eine Substitution heißt Grundsubstitution, wenn die Terme Konstanten sind. σ heißt konsistent in A, wenn jedes Auftreten einer Variablen mit dem gleichen Wert ersetzt wird. Werden alle Variablen von A (grund-)substituiert, dann ist $A\sigma$ eine (Grund-)Instanz von A.

Definition 2.1.18 (eval-Funktion)

1. *Der Wert einer gegebenen, grundinstanziierten, atomaren Formel L über einer gegebenen Faktenmenge $\mathcal{F}$ ist einer der Wahrheitswerte TRUE oder FALSE:*

$$eval(L,\mathcal{F}) \quad := \quad (\mathcal{F} \longrightarrow \{ TRUE \mid FALSE \}).$$

2. *Der Wert einer gegebenen Konjunktion grundinstanziierter Atome $L_1 \wedge \ldots \wedge L_n$ ist der Wahrheitswert der Konjunktion der Anwendung der eval-Funktion für jedes einzelne Literal L_i $(1 \leq i`n)$:*

$$eval(L_1 \wedge \ldots \wedge L_n,\mathcal{F}) \;:=\; eval(L_1,\mathcal{F}) \wedge \ldots \wedge eval(L_n,\mathcal{F}).$$

Die Anwendung von Deduktionsregeln wird durch einen konstruktiven Ableitungsoperator $T[R](\mathcal{F})$ ('immediate consequence operator') modelliert, der die durch eine Regel R unmittelbar ableitbaren Fakten definiert. Da in einer Datenbank nur wahre Fakten gespeichert oder ableitbar vorliegen, ist die Auswertung positiver Literale unproblematisch. Für die Auswertung negativer Literale wird die 'closed world assumption' (CWA) angenommen. Bei der CWA wird vorausgesetzt, daß ein negatives Fakt dann wahr ist, wenn es weder gespeichert noch ableitbar ist. Auf dieser Annahme über vollständiges Wissen basiert das operative Prinzip des 'negation as failure' (NAF). Liegt für ein negatives Literal das entsprechende positive Fakt nicht in der Datenbank vor, und ist es nicht durch Regeln ableitbar, dann wird geschlossen, daß das negative Literal wahr ist.

Für die simultane Anwendung einer Menge von Regeln wird auf der Basis von $T[R]$ der $T_{\mathcal{R}}$-Operator eingeführt. Da während seiner Anwendung die aktuell abgeleiteten Fakten noch nicht zugreifbar sind, wird der $T_{\mathcal{R}}$-Operator wiederholt auf einer kumulierten Faktenmenge ausgeführt. Die Faktenbasis einer $T_{\mathcal{R}}$-Anwendung besteht aus den Basisfakten $\mathcal{F}$ kumuliert mit den abgeleiteten Fakten der vorangegangenen $T_{\mathcal{R}}$-Anwendungen.

Definition 2.1.19 (Ableitungsoperator)
Sei $\mathcal{F}'$ eine beliebige Faktenmenge und $R := H \leftarrow B$ WITH $R_1^{help}, \ldots, R_m^{help}$ eine bereichsbeschränkte Datalog-Regel aus einer beliebigen Regelmenge $\mathcal{R}$.

1. $T[R](\mathcal{F}') := \{H\sigma \mid \sigma$ *ist eine konsistente Grundsubstitution für* B
   $\wedge\ eval(B\sigma, \mathcal{F}') = TRUE\}$.
2. $T_{\mathcal{R}}(\mathcal{F}') := \bigcup_{R \in \mathcal{R}} T[R](\mathcal{F}')$.
3. $T_{\mathcal{R}}{}^{\star}(\mathcal{F}') := \mathcal{F} \cup T_{\mathcal{R}}(\mathcal{F}')$.

Ist H ein parameterloses Kopfliteral, dann wird eine nicht leere Ergebnismenge des $T[R]$-Operators als TRUE interpretiert und die leere Menge als FALSE. Bei einer unkontrollierten Anwendung des $T_{\mathcal{R}}{}^{\star}$-Operators auf eine Regelmenge mit Negation kann es aufgrund verspäteter Ableitungen positiver Fakten zu unerwünschten Ableitungen kommen. Für eine stratifizierbare Regelmenge stellt die Ausführung des $T_{\mathcal{R}}{}^{\star}$-Operators gemäß der Stratifikationsebenen eine Möglichkeit dar, zu verhindern, daß negative Literale ausgewertet werden, bevor alle erforderlichen positiven Fakten abgeleitet wurden.

Definition 2.1.20 (Fixpunktsemantik deduktiver Datenbanken)

1. *Ist $\mathcal{D}^{\mathcal{D}}$ eine positive oder semi-positive DB. Dann ist die Bedeutung von $\mathcal{D}^{\mathcal{D}}$ der kleinste Fixpunkt $\mathcal{F}^{\star} := lfp\ (T_{\mathcal{R}}{}^{\star}(\mathcal{F}))$ von $T_{\mathcal{R}}{}^{\star}$, der $\mathcal{F}$ ganz enthält.*
2. *Ist $\mathcal{D}^{\mathcal{D}}$ eine stratifizierbare DB und λ eine Stratifikation von R mit n Ebenen $(n \geq 0)$, dann ist die iterierte Anwendung des $T_{\mathcal{R}}{}^{\star}$-Operators definiert als:*

   $$\begin{aligned} \mathcal{F}_0 &:= \mathcal{F} \\ \mathcal{F}_i &:= lfp\ (T_{\mathcal{R}}{}^{\star}(\mathcal{F}_{i-1})) \qquad mit\ 1 \leq i \leq n. \end{aligned}$$

   Die Bedeutung (iterierter Fixpunkt) von $\mathcal{D}^{\mathcal{D}}$ ist definiert als $\mathcal{F}^{\star} := \mathcal{F}_n$.

Definition 2.1.21 (Antwortmenge einer Anfrage)
Sei $\mathcal{R}'$ die um die Hilfsregeln einer Anfrage erweiterte Regelmenge $\mathcal{R}$ der DB $\mathcal{D}^{\mathcal{D}}$ und $\lambda = m$ eine Stratifikation von $\mathcal{R}'$. Die Anwendung einer Anfrage

$$\{(t_1, \ldots, t_n)\} \leftarrow B \ \ [WITH\ \ R_{Q1}^{help}, \ldots, R_{Ql}^{help}\ ]$$

mit $n \geq 0$ Parametern in der Zielliste liefert als Antwortmenge:

$$\{(t_1\sigma, \ldots, t_n\sigma) \mid \sigma\ \textit{ist eine Grundsubstitution für alle } X \in vars(B) \wedge\ eval(B\sigma,\ \mathcal{F}^{\star}) = TRUE\}.$$

wobei $\mathcal{F}^{\star} := \mathcal{F}_m$ *mit* $\mathcal{F}_i := lfp\ (T_{\mathcal{R}'}{}^{\star}(\mathcal{F}_{i-1})), \quad 1 \leq i \leq m.$

Diese Arbeit beschränkt sich auf die Auswertung stratifizierbarer Regelmengen, da in SQL auch nur stratifizierbare Regeln definierbar sind. Zu beachten ist, daß die Stratifikation (Definition 2.1.15) einer Regelmenge $\mathcal{R}$ auch die Hilfsregeln der WITH-Optionen von $\mathcal{R}$ berücksichtigt. Für die Berechnung des iterierten Fixpunkts $\mathcal{F}^{\star}$ von $\mathcal{D}^{\mathcal{D}}$, bzgl. dem eine

Anfrage ausgewertet wird, wird die Regelmenge $\mathcal{R}$ der Datenbank temporär um die Hilfsregeln der Anfrage ergänzt und die Stratifikation für die erweiterte Regelmenge ermittelt. Erst nach dieser Erweiterung wird $\mathcal{F}^\star$ berechnet und die Anfrage auf $\mathcal{F}^\star$ angewendet. Bei Anfragen mit parameterloser Zielliste wird die Ergebnismenge des $T[R]$-Operators mit dem leeren Tupel als TRUE interpretiert und die leere Menge als FALSE.

Definition 2.1.22 (Konsistente Faktenmenge)
Eine Faktenmenge $\mathcal{F}^\star$ heißt konsistent für eine Menge von Integritätsbedingungen $\mathcal{C}$, gdw.

$$\forall\ C_i \in \mathcal{C} \mid eval(C_i,\mathcal{F}^\star) = \mathit{TRUE}.$$

2.1.3 Änderungen und Transaktionen

In der Literatur hat sich bislang kein einheitliches Konzept einer Datenmanipulationssprache (DML) für Datalog durchgesetzt. Einen Einblick in verschiedene Änderungssprachen geben Cremers/Griefahn/Hinze in [CGH94] und Bonner/Kifer in [BK98]. Da diese DML-Konzepte sehr uneinheitlich sind, wird in diesem Abschnitt eine Datalog-DML definiert, die sich konzeptionell an der SQL-DML orientiert.

Grundsätzlich sind drei Änderungsarten bekannt: Einfügung, Löschung, Modifikation. Bei vielen Änderungssprachen sind aber nur Einfüge- und Löschanweisungen zulässig ([Abi88], [BBKL+88], [NK88], [MW88], [Oli91], [GMS93], [CW94], [Grie97]). Modifikationen werden bei diesen Sprachen als Einfügung des neuen und Löschung des alten Fakts durchgeführt. Nur wenige Sprachen wie z.B. SQL oder die DML von Olivé et.al. in [TO95], [UO94], [UO92] verfügen über eine Modifikationsanweisung.

Die einfachste Form einer Änderungsanweisung ist die primitive, bei der ein konkretes Fakt eingefügt, gelöscht oder modifiziert wird. Eine Menge von Fakten wird durch die Ausführung einer komplexen Änderungsanweisung manipuliert ('bulk update'). Der Qualifikationsteil komplexer Änderungsanweisungen ist eine Bedingung, die die zu manipulierenden Fakten erfüllen müssen. Im einfachsten Fall ist eine Manipulationsart je Anweisung erlaubt (SQL, DML aus [BMM92], ActLog aus [Grie97]). Bei anderen Änderungssprachen sind mehrere Manipulationsanweisungen in einem Ausdruck zulässig (Declarative Language $\mathcal{DL}$ [Abi88], RDL1 [BBKL+88], [KdMS90], Logic Data Language $\mathcal{LDL}$ [NK88], [NT89], Dynamic Prolog DLP [MW88], Transaction Logic $\mathcal{TR}$ [BK98], [BK95], Ultra [WFF98]).

Hinsichtlich der Verfügbarkeit prozeduraler Konstrukte wie z.B. bedingten Anweisungen, Sequenzen, Iterationen, etc. werden drei Sprachgruppen unterschieden:

- Eine **'rein' prozedurale DML** verfügt ausschließlich über primitive Änderungsanweisungen. Massenänderungen werden mittels Iteration formuliert wie in Transaction Language $\mathcal{TL}$ von Abiteboul/Vianu ([AV85], [AV87]).
- Eine **'rein' deklarative DML** verfügt über primitive und komplexe Änderungsanweisungen wie z.B. SQL ([ANSI99II]), $\mathcal{DL}$ von Abiteboul ([Abi88]), ActLog von Griefahn ([Grie97]) und die DMLs aus [CGH94], [BMM92].[5]

[5] Bei einigen Sprachen (SQL, DML aus [CGH94]) tritt der Sequenzbegriff im Zusammenhang mit der Ausführung einer Transaktion auf. Da das Transaktionskonzept grundsätzlich orthogonal zur eigentlichen DML ist, wird es hinsichtlich dieser Klassifizierung nicht berücksichtigt.

- Bei einer **deklarativen DML** mit einer **prozeduralen Steuerung** sind neben deklarativen Anweisungen für primitive und komplexe Änderungen auch prozedurale Steuerungsbefehle zulässig wie bei Prolog ([CGT90]), $\mathcal{TR}$, $\mathcal{LDL}$, RDL1, Ultra.

Die Semantik von Änderungssprachen wird u.a. nach einer deterministischen (mengenorientierten) oder nicht-deterministischen (tupelorientierten) Ausführung der Anweisungen klassifiziert. Bei einer deterministischen DML ([BMM92], [CGH94], [Grie97], SQL, $\mathcal{LDL}$, Ultra) werden klar zwei Phasen unterschieden. In der ersten werden alle zu ändernden Fakten ermittelt und in der zweiten die Menge der zu ändernden Fakten manipuliert. Bei einer nicht-deterministischen DML wie DLP, RDL1 werden die beiden Phasen für jedes von einer Änderungsoperation betroffene Fakt unmittelbar ausgeführt. Varianten der Sprachen $\mathcal{TL}$ und $\mathcal{DL}$ werden sowohl mit einer deterministischen wie auch nicht-deterministischen Semantik diskutiert.

Hinsichtlich des Determinismus einer DML muß als zweiter Sachverhalt die Transaktionssemantik berücksichtigt werden. Die Ausführung mehrerer Anweisungen im Rahmen einer Transaktion 'quasi simultan' als Menge (reihenfolgeunabhängig: ActLog, Ultra, [BMM92], [Ull89]) wird als deterministisch bezeichnet und die Ausführung als Sequenz von Anweisungen (reihenfolgeabhängig: SQL, Prolog, $\mathcal{TR}$, $\mathcal{TL}$, $\mathcal{DL}$, [CGH94]) als nicht-deterministisch. Diese beiden unterschiedlichen Konzepte werden auch als hypothetische (verzögerte) bzw. unmittelbare (reale) Ausführung bezeichnet. Die beiden Konzepte der Ausführung einer Änderungsanweisung bzw. einer Transaktion sind orthogonal zueinander. Hinsichtlich der Verfahren zur Propagierung von Folgeänderungen ist der (Nicht-) Determinismus einer Sprache selbst nur von untergeordneter Bedeutung. Er bestimmt nur welche Fakten geändert werden. Die Transaktionssemantik hingegen hat jedoch ganz essentiellen Einfluß auf den Ablauf eines Propagierungsverfahrens (siehe Kapitel 5).

Ziel dieser Arbeit ist nicht, eine vollständige Änderungssprache zu entwickeln. Die in diesem Kapitel entworfene Datalog-DML orientiert sich konzeptionell an der DML von SQL. Sie verfügt über primitive und komplexe Einfüge-, Lösch- und Modifikationsanweisungen. In Analogie zur SQL-DML aber auch analog zur deterministischen Ableitung von Fakten in Datalog wird die Ausführung von DML-Anweisungen ebenfalls deterministisch durchgeführt. Die Transaktionssemantik ist jedoch verschieden zu SQL. Sie ist deterministisch und orientiert sich an den in der Literatur zur Änderungspropagierung verwendeten Transaktionskonzepten. In Hinblick auf das Thema dieser Arbeit können Anwender nur Basisfakten ändern. Die Ausführung von DML-Operationen auf abgeleiteten Relationen ist nur in Folge von Basisfaktenänderungen zulässig und bleibt dem DB-System im Rahmen der Änderungspropagierung vorbehalten.

Syntax:

Bei deklarativen Änderungssprachen werden Anweisungen vielfach als Änderungsregeln dargestellt (ActLog-DML [Grie97], RDL1 [BBKL+88], $\mathcal{LDL}$ [NK88], [NT89]), wobei jedoch der imperative Charakter einer Änderungsanweisung gegenüber dem passiven Charakter von Deduktionsregeln nicht deutlich wird. Zur besseren Unterscheidung der Ausdrücke der Anfrage- und Änderungssprache beginnt der Qualifikationsteil einer Datalog-Änderungsanweisung analog zu SQL mit dem Schlüsselwort 'WHERE'. Um die Änderungsart auszudrücken, werden wie bei [Grie97] sogenannte dynamische Literale $+M$ und

$-M$ verwendet und für Modifikationen eine Tupelschreibweise $\pm(M^{new}, M^{old})$ oder abgekürzt $\pm M$.

Definition 2.1.23 (Änderungsanweisung)
Eine Änderungsanweisung (DML-Befehl) bzgl. einer DB $\mathcal{D}^{\mathcal{D}}$ ist induktiv definiert durch:

1. *Sind M, M^{new}, M^{old} grundinstanziierte DB-Atome und $\{+, -, \pm\}$ Änderungsoperatoren, dann sind folgende primitive Änderungsanweisungen spezifizierbar:*

Einfügung:	$+M(t_1, \ldots, t_n);$	*kurz:* $+M$
Löschung:	$-M(t_1, \ldots, t_n);$	*kurz:* $-M$
Modifikation:	$\pm(M(t_1^{new}, \ldots, t_n^{new}), M(t_1^{old}, \ldots, t_n^{old}));$	*kurz:* $\pm(M^{new}, M^{old})$.

2. *Sind M, M^{new}, M^{old} DB-Atome, $\{+, -, \pm\}$ Änderungsoperatoren und B eine Datalog-Formel, dann sind folgende bedingte (komplexe) Änderungsanweisungen spezifizierbar:*

Einfügung:	$+M$ *WHERE* B [*WITH* $R_1^{help}, \ldots, R_m^{help}$]
Löschung:	$-M$ *WHERE* B [*WITH* $R_1^{help}, \ldots, R_m^{help}$]
Modifikation:	$\pm(M^{new}, M^{old})$ *WHERE* B [*WITH* $R_1^{help}, \ldots, R_m^{help}$].

Die WITH-Option ist wahlfrei und bietet in Analogie zur Regeldefinition die Möglichkeit, Hilfsregeln zu definieren. Primitive Änderungsanweisungen sind Sonderfälle bedingter Anweisungen mit einer allgemeingültigen Bedingung im Qualifikationsteil.

Definition 2.1.24 (bereichsbeschränkte Änderungsanweisung)
Eine primitive Änderungsanweisung ist aufgrund ihrer Grundinstanziierung immer bereichsbeschränkt (sicher).
Eine bedingte Änderungsanweisung $+M$, $-M$, $\pm(M^{new}, M^{old})$ mit der Formel B im Qualifikationsteil ist bereichsbeschränkt (sicher), gdw. die analog definierte Deduktionsregel $M \leftarrow B$ bereichsbeschränkt ist.

Die Beschreibung einer Transaktion hängt wesentlich von der zugrunde liegenden Semantik der Ausführung mehrerer Änderungsanweisungen ab. Mit Blick u.a. auf SQL-basierte DB-Systeme definieren einige Autoren eine Transaktion als eine Folge von DML-Anweisungen (unmittelbare Ausführung) ([Ull89], [ANSI99I], [Date94], [AV85], [HS95], [CGH94], [Das92]). Vor dem Hintergrund der Theorie relationaler DB-Systemen beschreiben andere Autoren ([Ull89], [CW90], [BMM92], [Grie97], [WFF98]) eine Transaktion als eine Menge von DML-Anweisungen (verzögerte/hypothetische Ausführung). Die Qualifikationsteile aller Änderungsanweisungen werden dann auf dem alten DB-Zustand $\mathcal{F}^{old}$ ausgewertet. Erst nach dem Transaktionsende liegt die neue Faktenmenge $\mathcal{F}^{new}$ vor. Da den im Kontext von Datalog und Relationaler Algebra (RA) diskutierten Verfahren der Änderungspropagierung deterministische Transaktionskonzepte zugrunde liegen, wird für Datalog im weiteren mit einem analog definierten Transaktionsbegriff gearbeitet.

Unabhängig von dieser Diskussion muß eine Transaktion über die ACID-Eigenschaften verfügen ([HS95]). Eine Transaktion wird ganz oder gar nicht ausgeführt (**A**tomarität).

Am Ende einer Transaktion müssen alle Integritätsbedingungen erfüllt sein. Nur zwischenzeitlich können sie verletzt sein (Konsistenz (**C**onsistency)). Parallel ausgeführte Transaktionen können sich nicht beeinflussen[6] (**I**solation). Der (Netto-)Effekt einer erfolgreich durchgeführten Transaktion wird persistent gespeichert (**D**auerhaftigkeit).

Definition 2.1.25 (Transaktion)
Eine Menge $\mathcal{M} := \{M_1, \ldots, M_n\}$ *von primitiven oder bedingten Änderungen* $M_i \in \{+M, -M, \pm(M^{old}, M^{new})\}$, $(1 \leq i \leq n)$ *heißt eine Transaktion (kurz:* $\mathcal{M}$*).*

Semantik:

In diesem Abschnitt wird die Semantik von Basisfaktenänderungen definiert ohne Beachtung der Folgeänderungen bzgl. der abgeleiteten Relationen (unmittelbare Semantik)[7]. Bei der Definition der Semantik einer DML muß zwischen dem Effekt der Ausführung einer Änderungsanweisung und der Ausführung einer Menge von Änderungsanweisungen (Transaktion) unterschieden werden.

Für eine Änderungsanweisung wird angenommen, daß sie nur dann ausgeführt wird, wenn dadurch die Faktenmenge effektiv verändert wird. Zu diesem Zweck werden vom DB-System vor der Ausführung Effektivitätstests durchgeführt.

1. Die Einfügung eines Fakts $p(a)$ in $\mathcal{F}^{old}$ ist nur dann effektiv, wenn $p(a) \notin \mathcal{F}^{old}$.
2. Die Löschung eines Fakts $p(a)$ in $\mathcal{F}^{old}$ ist nur dann effektiv, wenn $p(a) \in \mathcal{F}^{old}$.
3. Die Modifikation eines Fakts $p(a, b)$ nach $p(a, c)$ in $\mathcal{F}^{old}$ ist nur dann effektiv, wenn $p(a, b) \in \mathcal{F}^{old}$ und $p(a, c) \notin \mathcal{F}^{old}$.

Für die Ausführung nur effektiver Änderungen wird der Regelrumpf einer Änderungsanweisung DBS-intern um Effektivitätstests ($\wedge A\sigma$ bzw. $\wedge\neg A\sigma$) erweitert. Die Idee der 'effective manipulation'[8] ist bereits bei Küchenhoff in [Kue91] für Einfügungen und Löschungen zu finden, ohne daß sie dort formal spezifiziert ist. Die Frage nach der (Nicht-) Ausführung effektloser Änderungsanweisungen ist zudem von besonderer Bedeutung für die Ausführung von Triggern in aktiven DB-Systemen. Aktive Regeln, die bzgl. DML-Ereignissen feuern und wiederum DML-Anweisungen im Aktionsteil enthalten, erzeugen dann u.U. 'Folgeänderungen' auf der Basis nicht effektiver Änderungen.

Definition 2.1.26 (Menge effektiv betroffener Tupel)
Sei $\mathcal{F}$ *eine Basisfaktenmenge und* $M \in \{+M, -M, \pm(M^{new}, M^{old})\}$ *eine Änderungsanweisung mit dem Qualifikationsteil* B *und dem Relationssymbol* $R \in \mathcal{PS}$ *im Kopfliteral.*

1. $\mathcal{T}(M, \mathcal{F})$ *ist die Menge der durch die Anwendung von* M *auf* $\mathcal{F}$ *betroffenen Tupel.*

$$\mathcal{T}(M, \mathcal{F}) := \langle \mathcal{T}^-(M, \mathcal{F}), \mathcal{T}^+(M, \mathcal{F}), \mathcal{T}^\pm(M, \mathcal{F}) \rangle.$$

[6] In Hinblick auf das Thema der Arbeit ist diese Eigenschaft nicht von Relevanz.

[7] Die Semantik induzierter Änderungen wird in Kapitel 4 detailliert diskutiert.

[8] Küchenhoff verwendet den Begriff 'effective updates'. Da in SQL der Begriff UPDATE für die Modifikationsanweisung reserviert ist, wird hier von effektiven Manipulationen gesprochen.

$$\mathcal{T}^+(+M,\mathcal{F}) := \{R\sigma \mid eval(B\sigma \wedge \neg R\sigma, \mathcal{F}) = TRUE\}$$
$$\mathcal{T}^-(+M,\mathcal{F}) := \emptyset.$$
$$\mathcal{T}^\pm(+M,\mathcal{F}) := \emptyset.$$

$$\mathcal{T}^+(-M,\mathcal{F}) := \emptyset.$$
$$\mathcal{T}^-(-M,\mathcal{F}) := \{R\sigma \mid eval(B\sigma \wedge R\sigma, \mathcal{F}) = TRUE\}.$$
$$\mathcal{T}^\pm(-M,\mathcal{F}) := \emptyset.$$

$$\mathcal{T}^+(\pm(M^{new},M^{old}),\mathcal{F}) := \emptyset.$$
$$\mathcal{T}^-(\pm(M^{new},M^{old}),\mathcal{F}) := \emptyset.$$
$$\mathcal{T}^\pm(\pm(M^{new},M^{old}),\mathcal{F}) := \{ (R^{new}\sigma, R^{old}\sigma) \mid eval(B\sigma \wedge \neg R^{new}\sigma \wedge R^{old}\sigma, \mathcal{F}) = TRUE \}.$$

2. *Ist* $\mathcal{M} := \{M_1,\ldots,M_n\}$ *eine Transaktion, dann sind die folgenden* $\mathcal{T}$*-Mengen die Mengen einzufügender/zu löschender/zu modifizierender Tupel.*

$$\mathcal{T}(\mathcal{M},\mathcal{F}) := \langle \mathcal{T}^+(\mathcal{M},\mathcal{F}), \mathcal{T}^-(\mathcal{M},\mathcal{F}), \mathcal{T}^\pm(\mathcal{M},\mathcal{F}) \rangle.$$

$$\mathcal{T}^+(\mathcal{M},\mathcal{F}) := \bigcup_{i=1}^n \mathcal{T}^+(M_i,\mathcal{F}).$$
$$\mathcal{T}^-(\mathcal{M},\mathcal{F}) := \bigcup_{i=1}^n \mathcal{T}^-(M_i,\mathcal{F}).$$
$$\mathcal{T}^\pm(\mathcal{M},\mathcal{F}) := \bigcup_{i=1}^n \mathcal{T}^\pm(M_i,\mathcal{F}).$$
$$\mathcal{T}^{\pm+}(\mathcal{M},\mathcal{F}) := \{ M^{new} \mid (M^{new},M^{old}) \in \mathcal{T}^\pm(\mathcal{M},\mathcal{F}) \}.$$
$$\mathcal{T}^{\pm-}(\mathcal{M},\mathcal{F}) := \{ M^{old} \mid (M^{new},M^{old}) \in \mathcal{T}^\pm(\mathcal{M},\mathcal{F}) \}.$$

Die Semantik einer Transaktion (Menge von Basisfaktenänderungen) wird im wesentlichen durch drei Grundannahmen bestimmt.

- Semantik der Ausführung einer Änderungsanweisung:
  Nur effektiver Basisfaktenänderungen werden ausgeführt.

- (nicht-)deterministische Ausführung einer Menge von Anweisungen:
  Für eine Transaktion im Datalog-Kontext wird eine deterministische Ausführung vorausgesetzt.

- Regeln der Nettoeffektberechnung:
  Die Nettoeffektberechnung ist insbesondere notwendig bei deterministischen DML-Sprachen mit einer deterministischen Transaktionssemantik, wie sie hier für Datalog spezifiziert wird.

Die 'quasi simultane' Ausführung einer Menge von Basisfaktenänderungen zum Transaktionsende kann eine Vielzahl von Konfliktfällen in sich bergen, die auf Abhängigkeiten zwischen den betroffenen Fakten basieren. Zur Handhabung dieser Konflikte[9] müssen Regeln definiert werden. Als Ergebnis der Anwendung dieser Regeln (Nettoeffektberechnung) liegt der Nettoeffekt einer Menge von Änderungen vor.

[9] In SQL stellen diese Fälle aufgrund der Ausführungsreihenfolge keine Konflikte dar, wenn effektive Änderungen ausgeführt werden. Folgen von Änderungen können jedoch wie z.B. in den Fällen 3 und 6 zu einer Anweisung zusammengefaßt werden (SQL Abschnitt 2.2.4)

1. Ein Fakt wird in einer Transaktion auf die gleiche Art mehrfach manipuliert. $\{+p(a)\ +p(a)\}$

2. Ein Fakt wird in einer Transaktion sowohl eingefügt als auch gelöscht. $\{+p(a)\ -p(a)\}$

3. Ein Fakt, das dem alten Wert eines modifizierten Fakts entspricht, wird eingefügt. $\{+p(a,b)\ \pm(p(a,c),p(a,b))\}$

4. Ein Fakt, das dem neuen Wert eines modifizierten Fakts entspricht, wird eingefügt. $\{+p(a,c)\ \pm(p(a,c),p(a,b))\}$

5. Ein Fakt, das dem alten Wert eines modifizierten Fakts entspricht, wird gelöscht. $\{-p(a,b)\ \pm(p(a,c),p(a,b))\}$

6. Ein Fakt, das dem neuen Wert eines modifizierten Fakts entspricht, wird gelöscht. $\{-p(a,c)\ \pm(p(a,c),p(a,b))\}$

7. Ein Fakt wirt derart modifiziert, daß es dem alten Wert einer anderen Modifikation entspricht (Folgemodifikationen). $\{\pm(p(a,d),p(a,c)\ \pm(p(a,c),p(a,b))\}$

8. Mehrere Fakten werden zu dem gleichen neuen Wert modifiziert. $\{\pm(p(a,d),p(a,c))\ \pm(p(a,d),p(a,b))\}$

9. Ein Fakt wird durch mehrere Modifikationen in verschiedene neue Werte geändert. $\{\pm(p(a,d),p(a,b))\ \pm(p(a,c),p(a,b))\}$

Der erste Fall kann dadurch ausgeschlossen werden, daß im Rahmen der Auswertung einer Änderungsanweisung eine Menge $\mathcal{T}$ und keine Multimenge ('bag') von zu ändernden Fakten ermittelt wird. Ebenso ist eine Transaktion eine Menge von Änderungen. Für effektive Basisfaktenänderungen stellen die Fälle 2, 3, 6 und 7 sich gegenseitig ausschließende Änderungen dar. Aufgrund des Effektivitätstests, der bei der Auswertung der Anweisungen auf der Faktenmenge $\mathcal{F}^{old}$ durchgeführt wird, ist entweder die eine oder die andere Anweisung nicht ausführbar und hat somit keinen Effekt, der im Rahmen der Nettoeffektberechnung berücksichtigt werden müßte. Diese Konfliktfälle können bei effektiven Manipulationen nicht auftreten. Von den sich gegenseitig subsumierenden Änderungen der Fälle 4 und 5 reicht es aus, nur die Änderung beim Nettoeffekt zu berücksichtigen, die hinsichtlich ihrer Konsequenzen die andere Manipulation beinhaltet: Modifikation beinhaltet die Löschung des alten und Einfügung des neuen Fakts. Die Modifikationen der Fälle 8 und 9 stellen keine echten Konflikte dar sondern nur Änderungen, deren Einzelanweisungen Einfügung und Löschung teilweise redundant sind. Da erwartet wird, daß die Propagierung von Modifikationen häufig einen Effizienzgewinn darstellt, werden diese 'teilweise' redundanten Modifikationen nicht aufgespalten, sondern unverändert in die Ergebnismenge des Nettoeffekts aufgenommen. Nach Anwendung dieser Nettoeffektregeln ist eine Menge von Änderungen konfliktfrei und kann reihenfolgeunabhängig am Ende einer Transaktion ausgeführt werden.

Bei anderen Manipulationssprachen sind diese Probleme der Nettoeffektberechnung ganz anders gelöst worden, mit zum Teil sehr verschiedenen Folgezuständen. Da nur wenige Sprachen über eine Modifikationsanweisung verfügen, gestaltet sich bei den anderen Sprachen die Nettoeffektberechnung wesentlich einfacher, da weniger Konfliktfälle

auftreten können (nur Fall 1 und 2). Eine oft verwendete Annahme zur Lösung des Konfliktfalls 2 ist die, eine Reihenfolge zu definieren (nicht-deterministische (prozedurale) Transaktionssemantik), z.B. wie bei [LST86], [BMM92], [CW94], [GL95], [CGLMT96], wo Löschungen vor Einfügungen auszuführen sind.[10]

$\mathcal{F}^{old}$	Transaktion	Nettoeffekt	$\mathcal{F}^{new}$
$p(a)$	$\{+p(a)\ -p(a)\}$	$\{-p(a),\ +p(a)\}$	$p(a)$
$\emptyset$	$\{+p(a)\ -p(a)\}$	$\{-p(a),\ +p(a)\}$	$p(a)$.

Griefahn in [Grie97] und Ceri/Widom/Finkelstein in [WF90], [CW90] behandeln Löschungen und Einfügungen symmetrisch. Bei konkurrierenden Änderungen eines Fakts wie in Fall 2 werden beide nicht ausgeführt. Für das Konfliktbeispiel heißt dies:

$\mathcal{F}^{old}$	Transaktion	Nettoeffekt	$\mathcal{F}^{new}$
$p(a)$	$\{+p(a)\ -p(a)\}$	$\{\}$	$p(a)$
$\emptyset$	$\{+p(a)\ -p(a)\}$	$\{\}$	$\emptyset$.

Bei der DML Ultra ([WFF98]) wird auf diesen Fehlerfall einer sogenannten 'nichtkonsistenten' Änderung mit dem Abbruch der Transaktion reagiert.

Neben der Nichtausführung von Anweisungen kann die Kombination von Ereignissen sinnvoll sein oder deren Aufspaltung. Die weiteren Beispiele (Fälle 4, 6 und 9) richten sich nach den Vorgaben in [WF90], [CW90] und vermitteln einen Eindruck von möglichen Interpretationen:[11]

$\mathcal{F}^{old}$	Transaktion	Nettoeffekt	$\mathcal{F}^{new}$
$p(a,b)$	$\{\pm(p(a,c),p(a,b))\ -p(a,c)\}$	$\{-p(a,b)\}$	$\emptyset$
$\emptyset$	$\{\pm(p(a,c),p(a,b))\ +p(a,b))\}$	$\{+p(a,c)\}$	$p(a,c)$
$p(a,b)$	$\{\pm(p(a,c),p(a,b))\ \pm(p(a,d),p(a,c))\}$	$\{\pm(p(a,d),p(a,b))\}$	$p(a,d)$.

Diese Regeln implizieren eine Ausführungsreihenfolge und sind daher nicht geeignet für die hier vorgestellte Spezifikation einer deterministischen Transaktionssemantik.

Die ebenfalls vielfach anzutreffende und auf den ersten Blick sehr intuitive Annahme, Modifikationen in Einfügungen und Löschungen aufzuspalten ($\mathcal{TR}$, Dynamic Prolog, $\mathcal{LDL}$, [Dec86], [LST86], [Oli91], [CW94], [Kue91], [Grie97], u.v.m.) impliziert, daß die Eigenschaft der atomaren Ausführung einer Modifikation verloren geht. Einfügung des neuen und Löschung des alten Fakts erfolgen unabhängig voneinander:[12]

$\mathcal{F}^{old}$	Transaktion	Nettoeffekt	$\mathcal{F}^{new}$
$p(a,b)$	$\{\pm(p(a,c),p(a,b))\}$	$\{+p(a,c)\ -p(a,b)\}$	$p(a,c)$
$\emptyset$	$\{\pm(p(a,c),p(a,b))\}$	$\{+p(a,c)\ -p(a,b)\}$	$p(a,c)$.

[10] Bei der hier vorgestellten Sprache wird aufgrund der Effektivitätsprüfung vor der Nettoeffektberechnung für mindestens eine der beiden Änderungen die Effektlosigkeit erkannt. Der Konfliktfall kann nicht eintreten.

[11] Da auch bei diesen Beispielen die Effektivitätsprüfung bei der hier vorgestellten Sprache für mindestens eine der beiden Manipulationen Effektlosigkeit anzeigen würde, können diese drei Konfliktfälle nicht eintreten. Im ersten und dritten Beispiel hätte nur die Modifikation von $p(a,b)$ nach $p(a,c)$ einen Effekt und im zweiten nur die Einfügung $p(a,b)$.

[12] Im zweiten Beispiel, wo das alte Fakt $p(a,b)$ nicht in $\mathcal{F}^{old}$ enthalten ist, wird bei der hier vorgestellten Sprache aufgrund des Effektivitätstests die Modifikation nicht ausgeführt und $\mathcal{F}^{new} = \emptyset$.

Im Rahmen der Diskussion von Verfahren der Änderungspropagierung ist vielfach der Ansatz zu finden ([Dec86], [Oli91], [UO92], [TO95], [Kue91]), bei dem der Nettoeffekt einer Transaktion anhand der Unterschiede zwischen der neuen und der alten Faktenmenge $\mathcal{F}^{new}$ und $\mathcal{F}^{old}$ ermittelt wird. Die eigentliche Transaktion, die diesen Nettoeffekt verursacht hat, wird nicht weiter berücksichtigt. Die Autoren tragen damit dem Umstand Rechnung, daß die 'unmittelbare Semantik' der Transaktion zwar bestimmt, welche geänderten Basisfakten zu propagieren sind, aber keinen Einfluß darauf nimmt, wie sie zu propagieren sind.

Da von den Konfliktfällen der Nettoeffektberechnung bereits die Fälle 1-3, 6 und 7 aufgrund der Ausführung nur effektiver Änderungen gelöst sind (Definition 2.1.26) und die Fälle 8, 9 keine Konflikte darstellen, müssen im Rahmen der eigentlichen Nettoeffektberechnung nur noch die subsumierenden Änderungen der Fälle 4 und 5 berücksichtigt werden.

Definition 2.1.27 (Nettoeffekt einer Menge von Änderungen)
Sei $\mathcal{F}$ die Basisfaktenmenge von $\mathcal{D}^{\mathcal{D}}$ und $\mathcal{M}$ eine Transaktion, deren Anwendung die Menge $\mathcal{T}(\mathcal{M},\mathcal{F}) := \langle \mathcal{T}^{+}(\mathcal{M},\mathcal{F}), \mathcal{T}^{-}(\mathcal{M},\mathcal{F}), \mathcal{T}^{\pm}(\mathcal{M},\mathcal{F})\rangle$ mit den effektiv zu ändernden Fakten liefert. Dann ergeben sich für die Ausführung von $\mathcal{M}$ folgende Nettoeffekte.

1. *Nettoeffekt der Einfügungen:* $\Delta^{+}(\mathcal{M},\mathcal{F}) := \mathcal{T}^{+}(\mathcal{M},\mathcal{F}) \setminus \mathcal{T}^{\pm+}(\mathcal{M},\mathcal{F})$
   Nettoeffekt der Löschungen: $\Delta^{-}(\mathcal{M},\mathcal{F}) := \mathcal{T}^{-}(\mathcal{M},\mathcal{F}) \setminus \mathcal{T}^{\pm-}(\mathcal{M},\mathcal{F})$
   Nettoeffekt der Modifikationen: $\Delta^{\pm}(\mathcal{M},\mathcal{F}) := \mathcal{T}^{\pm}(\mathcal{M},\mathcal{F})$

2. $\Delta^{\pm+}(\mathcal{M},\mathcal{F}) := \{\ M^{new} \mid (M^{new}, M^{old}) \in \Delta^{\pm}(\mathcal{M},\mathcal{F})\ \}$
   $\Delta^{\pm-}(\mathcal{M},\mathcal{F}) := \{\ M^{old} \mid (M^{new}, M^{old}) \in \Delta^{\pm}(\mathcal{M},\mathcal{F})\ \}$

3. *Der Nettoeffekt einer Menge von Änderungsanweisungen $\mathcal{M}$ wird vollständig definiert durch die Δ-Mengen:*

$$\Delta(\mathcal{M},\mathcal{F}) := \langle\ \Delta^{+}(\mathcal{M},\mathcal{F}), \Delta^{-}(\mathcal{M},\mathcal{F}), \Delta^{\pm}(\mathcal{M},\mathcal{F})\ \rangle\ .$$

Ist der Bezug zur ausgeführten Menge von Änderungsanweisungen $\mathcal{M}$ und zur Basisfaktenmenge $\mathcal{F}$ eindeutig, so kann die Kurzschreibweise $\Delta := \langle\ \Delta^{+}, \Delta^{-}, \Delta^{\pm}\ \rangle$ verwendet werden. Der Nettoeffekt einer Transaktion definiert den Zustandsübergang von der alten Basisfaktenmenge $\mathcal{F}^{old}$ zu der neuen $\mathcal{F}^{new}$ nach Ausführung der Transaktion vollständig. Die Δ-Mengen können deterministisch auf $\mathcal{F}^{old}$ ausgeführt werden, denn sie sind disjunkte Faktenmengen (Konfliktfreiheit).

Aus der Semantik einer Transaktion leitet sich auch das Integritätsprüfungsverhalten ab. Bei einer deterministischen Transaktion können alle betroffenen Integritätsbedingungen geprüft werden, nachdem alle Änderungen ausgeführt wurden und der neue DB-Zustand vorliegt. Diese Vorgehensweise ist in der Literatur weit verbreitet bei der Diskussion von Änderungspropagierungsverfahren (u.a. in [Oli91], [CW91], [GMS93], [GL95], [Grie97]). In Analogie zu den DML-Anweisungen bzgl. Basisfakten werden hier im Rahmen der Integritätsprüfung auch nur diejenigen Integritätsbedingungen berücksichtigt, die sich ausschließlich auf Basisrelationen beziehen ($rel_{idb}(C_i) = \emptyset$).

Definition 2.1.28 (Konsistenter Zustand $\mathcal{F}^{new}$)
$\mathcal{C}$ sei die Menge der Integritätsbedingungen von $\mathcal{D}^{\mathcal{D}}$ und $\mathcal{F}^{old}$ sei eine konsistente Basisfaktenmenge, auf der die Transaktion $\mathcal{M}$ mit dem Nettoeffekt $(\Delta(\mathcal{M}, \mathcal{F}^{old})$ ausgeführt wird ($\Delta(\mathcal{M}, \mathcal{F}^{old}) := \langle \Delta^+(\mathcal{M}, \mathcal{F}^{old}),\ \Delta^-(\mathcal{M}, \mathcal{F}^{old}),\ \Delta_D^{\pm}(\mathcal{M}, \mathcal{F}^{old}) \rangle$).
Die Ausführung von $\mathcal{M}$ überführt den alten konsistenten Zustand $\mathcal{F}^{old}$ in einen neuen konsistenten (End-)Zustand $\mathcal{F}^{new}$.

$$\mathcal{F}^{new} := \begin{cases} \mathcal{F}^{old} \cup (\Delta^+(\mathcal{M}, \mathcal{F}^{old}) \cup \Delta^{\pm+}(\mathcal{M}, \mathcal{F}^{old})) & falls\ \forall\ C_i \in \mathcal{C} \mid \\ \quad \setminus (\Delta^-(\mathcal{M}, \mathcal{F}^{old}) \cup \Delta^{\pm-}(\mathcal{M}, \mathcal{F}^{old})) & eval(C_i, \mathcal{F}^n) = TRUE \\ \mathcal{F}^{old} & sonst \end{cases}$$

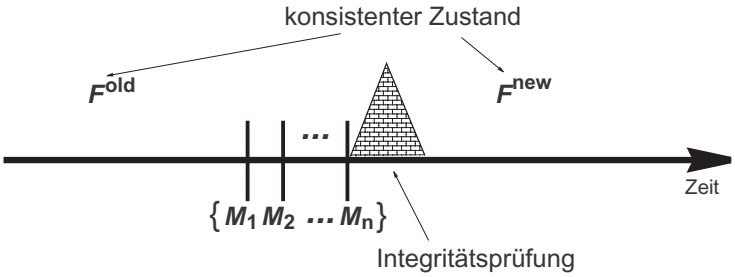

Abbildung 2.2: Datalog-Transaktion

In Abbildung 2.2 wird das Datalog-Transaktions- und Integritätskonzept veranschaulicht, so wie es dieser Arbeit zugrunde gelegt wird. Im Rahmen einer Transaktion wird deterministisch eine Menge von Änderungen ausgeführt und geprüft, wodurch ein konsistenter Zustand in einen wiederum konsistenten Zustand überführt wird. Obwohl es diesbezüglich in Datalog keine Quasi-Standardisierung gibt, wird in Abgrenzung zu SQL im weiteren die Bezeichnung Datalog-Transaktions- und Integritätskonzept verwendet. Sind aus dem Zusammenhang sowohl die Transaktion $\mathcal{M}$ wie auch die Ausgangsfaktenmenge $\mathcal{F}^{old}$, bzgl. der $\mathcal{M}$ ausgeführt wird, offensichtlich, so wird für den konsistenten (End-)Zustand $\mathcal{F}^n(\mathcal{F}^{old}, \mathcal{M})$ nur die Kurzbezeichnung $\mathcal{F}^{new}$ verwendet.

2.1.4 Trigger

Auch für Triggersysteme hat sich bislang kein einheitliches und allgemein anerkanntes Konzept durchgesetzt. Umfassende Einführungen in die Thematik aktiver Regeln werden in [DG96], [HS95] gegeben. Eine über die Grundlagen hinausgehende ausführliche Beschreibung und Klassifizierung verschiedener aktiver Regel-Systeme sind u.a. in [Grie97], [Rei96], [WC96], [Pat99], [HW93] zu finden. Reinert untersucht in [Rei96] die Eignung verschiedener aktiver Regelsysteme als Basis für die Implementierung eines Integritätskonzepts und analysiert detailliert die Ausführungsmodelle von Regelsystemen wie Ariel, Alert, CA-Ingres, Heraclitus, HiPAC, Ode, POSTGRES, SAMOS, Starburst oder die Ansätze von Hammer/McLeod und Eswaran. Ein guter Einblick in verschiedene

Einsatzmöglichkeiten für aktive Regeln wird in [CCW00] gegeben. Als ein klassisches Aufgabengebiet sehen Ceri et.al. die prototypische Implementierung systemgenerierter Trigger zur Unterstützung von Kernfunktionen des DB-Systems wie z.B. Integritätsprüfung und materialisierte Sichten.

Grundlegende Eigenschaft eines Trigger ist, daß seine Aktionen automatisch ausgeführt werden, wenn ein bestimmtes Ereignis eintritt (EA-Paradigma). Die Triggerkomponente als Bestandteil des DB-Systems erkennt das Auftreten eines Ereignisses im Sinne des Regelkonzepts (Datenmanipulationen, Zeitpunkte, abstrakte Ereignisse etc.) und ermittelt die von diesem Ereignis betroffenen Trigger. Zu einem bestimmten Zeitpunkt (unmittelbar, verzögert) werden die Aktionen des Triggers (Anfragen, Änderungsanweisungen, Transaktionssteuerung (COMMIT), Prozeduraufrufe, Erzeugen abstrakter Ereignisse) ausgeführt. Bei Triggerkonzepten, die dem ECA-Paradigma genügen, können für Trigger Bedingungen (C) definiert werden, die zum Zeitpunkt der Aktivierung bzw. Ausführung angewendet werden. Ist die Bedingung nicht erfüllt, so wird der Trigger nicht ausgeführt.

Eine weitere sehr wesentliche Eigenschaft ist die Granularität der Regelausführung. Wird für eine Änderungsanweisung unabhängig davon, wie viele Fakten manipuliert werden, ein Trigger nur einmal gefeuert, so die Regelausführung mengenorientiert bezeichnet. Feuert ein Trigger hingegen für jedes geänderte Fakt, so wird sie als instanzorientiert bezeichnet. Da aktive Regelsysteme in dieser Arbeit nur als Implementierungshilfen für verschiedene Verfahren verwendet werden, wird auf eine ausführliche Diskussion aller Eigenschaften verzichtet.

Griefahn hat in [Grie97] die Eigenschaften von Triggersystemen ausführlich analysiert und ein detailliertes Klassifikationsschema entwickelt. Durch die Einordnung bekannter Trigger-Systeme wie Starburst, Ariel, Postgres, A-RDL, HiPAC, Ode, Chimera in dieses Klassifikationsschema hat sie die Semantik dieser Systeme überschaubar und gut vergleichbar spezifiziert. Ausgehend von den Vor- und Nachteilen der bekannten Systeme hat sie speziell für den Einsatz zur Implementierung eines inkrementellen Änderungspropagierungsverfahrens die ActLog-Trigger entwickelt.

ActLog-Trigger:

Griefahn hat in [Grie97] ein Triggerkonzept insbesondere hinsichtlich der Anwendung für die Implementierung von Änderungspropagierungsverfahren definiert.

Definition 2.1.29 (ActLog-Trigger)
Ein ActLog-Trigger ist eine EA-Regel der Form

$$G \ : \ \star M \quad \Rightarrow \quad *M',$$

wobei G das Triggersymbol ist. $\star M$ ist das Ereignissymbol und $*M'$ die Triggeraktion. $\star M$ und $*M'$ sind jeweils ActLog-Änderungsanweisungen ($\star, * \in \{+, -\}$).

Für einen Trigger kann auch eine Liste disjunktiver Ereignisse bzw. eine Folge von Aktionen definiert werden

$$G \ : \ \{ \ \star M_1, \ \ldots, \ \star M_n \ \} \quad \Rightarrow \quad *M_{n+1}, \ \ldots, \ *M_{n+m}.$$

Trigger feuern für die Ausführung von effektiven Einfügungen und Löschungen von Basisfakten ('safe events'), und als Aktionen sind ebenfalls Einfügungen und Löschungen zulässig. Die ActLog-DML entspricht im wesentlichen der in Abschnitt 2.1.3 definierten Datalog-DML. ActLog-Trigger können für verschiedene Ereignisse auf verschiedenen Basisrelationen definiert werden. ActLog-Trigger müssen sichere Trigger sein. Alle freien Variablen im Aktionsteil $(*M', *M_{n+1}, .., *M_{n+m})$ müssen im Ereignisteil gebunden sein. Zwischen Triggern können partielle Ordnungen definiert werden. Die Prioritätendeklaration $G_1 \ll G_2$ definiert den Vorrang von Trigger G_1 vor G_2 bei der Ausführung.

Eine exakte zeitliche Bindung zwischen dem Aktivieren eines Triggers und seiner Ausführung wird für ActLog nicht definiert. Für eine Anwendung der Trigger im Rahmen einer Anfrageauswertung ist eine unmittelbare Ausführung erforderlich, im Rahmen einer Integritätsprüfung eine zum Transaktionsende verzögerte Ausführung. So wie Griefahn die Trigger für Änderungspropagierungsverfahren zum Transaktionsende verwendet, und aufgrund des Triggerausführungsmodells mit der Berechnung des Nettoeffekts der Änderungen zu Beginn jeder Iterationsrunde ist diese Fragestellung unerheblich.

Zu einem Zeitpunkt (Transaktionsende) werden alle seit dem letzten Triggerausführungszeitpunkt aufgetretenen Ereignisse mengenorientiert (deterministisch) bearbeitet. Die Trigger selbst werden instanzorientiert (tupelorientiert) ausgeführt. Betrifft eine Änderungsanweisung effektiv mehrere Fakten, so wird für jedes manipulierte Fakt ein Trigger gefeuert. Ereignis und gefeuerter Trigger werden zu einer Ausführungseinheit zusammengefaßt, dem 'rule-event pair'. Aufgrund gemeinsamer Variablen der Literale im Ereignis- und Aktionsteil (sicherer Trigger) werden Variablenbindungen für die auszuführenden Aktionen erzeugt. Trigger-Ereignispaare werden nur ausgeführt, wenn zum Zeitpunkt ihrer Ausführung das feuernde Ereignis noch sicher ('safe') ist. Ein Ereignis ist z.B. nicht mehr sicher, wenn ein kompensierendes Ereignis eingetreten ist, wie z.B. bei Einfügungen und Löschungen des gleichen Fakts (Nettoeffektberechnung).

Die Ausführung der durch eine Menge von Ereignissen aktivierten Trigger-Ereignispaare erfolgt iterativ. Jeder Iterationsschritt ist in drei Phasen unterteilt: 'triggering phase', 'selection phase', 'execution phase'. Die Schritte werden solange wiederholt, bis die Menge der Trigger-Ereignispaare leer ist.

1. Aktivierung ('triggering phase'): Aus der Menge der in der Iterationsrunde i auszuführenden Trigger-Ereignispaare werden die Trigger-Ereignispaare entfernt, die in der letzten Iterationsrunde $i-1$ ausgeführt wurden (ereigniskonsumierend). Aus der Menge der verbliebenen Trigger-Ereignispaare werden zudem die Paare entfernt, deren Ereignis nicht mehr 'safe' ist (Nettoeffektberechnung). Für alle Ereignisse seit der letzten Iterationsrunde $i-1$ werden Trigger-Ereignispaare gebildet und in die Menge der auszuführenden Trigger-Ereignispaare aufgenommen.

2. Selektion ('selection phase'): In einer Iterationsrunde i werden die Trigger-Ereignispaare mit der maximalen Priorität ausgeführt. Die Priorität von Trigger-Ereignispaaren ist maximal, gdw. zum aktuellen Zeitpunkt keine anderen Paare aktiviert sind, deren Priorität höher ist.

3. Ausführung ('execution phase'): Die Menge der in einer Iterationsrunde i auszuführenden Trigger-Ereignispaare wird deterministisch in drei Schritten auf dem DB-Zu-

stand $\mathcal{F}^{i-1}$ der vorangegangenen Iterationsrunde $i-1$ ausgeführt. Zuerst wird die Menge der mit den aktuellen Werten der Parameter des feuernden Ereignisses instanziierten Änderungsanweisungen ermittelt. Die Formeln dieser Änderungsanweisungen werden auf $\mathcal{F}^{i-1}$ angewendet. Abschließend werden die Änderungen deterministisch auf $\mathcal{F}^{i-1}$ ausgeführt.

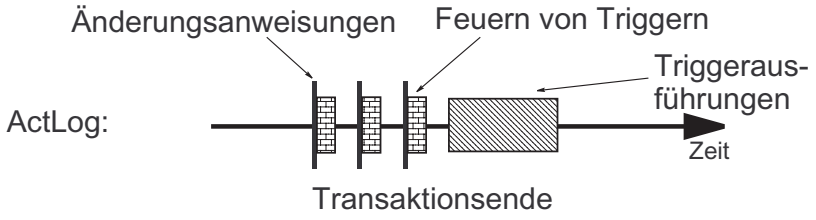

Abbildung 2.3: ActLog-Triggerausführungsmodell

In Abbildung 2.3 sind die zeitlichen Abläufe der Aktivierung und Ausführung von ActLog-Triggern skizziert.

2.2 SQL

In diesem Abschnitt werden die Anfrage- und Sichtkonzepte sowie die Integritäts- und DML-Konzepte von SQL3 ausführlich diskutiert. Aufgrund der vielfachen Verwendung von SQL bzw. SQL-Dialekten als Sprachen für relationale DB-Systeme kommerzieller Hersteller gibt es eine Vielzahl von Publikationen zum Thema SQL. Gulutzan/Pelzer spezifizieren in [GP99] den neuen Standard SQL3 anschaulich und mit vielen Praxisbeispielen. Date/Darwen definieren in [DD97] SQL2 in sehr übersichtlicher Form. Sie gehen dabei formaler und weniger beispielorientiert vor als andere Autoren. Groff/Weinberg erläutern SQL2 in [GW99] hingegen anhand vieler praxisnaher Beispiele. Zudem geben sie einen Überblick über Abweichungen verschiedener Implementierungen von den Vorgaben des Standards, so z.B. für DB2 V5.2, Oracle V8.0.4, Microsoft SQL Server 7.0, Informix 7.22.TC2 und viele mehr. Celko vermittelt in [Cel00] einen umfassenden Einblick in SQL2 mit vielen Tips zur performanten Programmierung. Melton/Simon geben [MS93] für Anwendungsentwickler einen praxisnahen und guten Einblick in SQL2. Im weiteren orientiere ich mich im wesentlichen direkt an den Publikationen des Standardisierungsgremiums zum SQL3-Standard[13] ([ANSI99I], [ANSI99II]) bzw. an [GP99] und soweit SQL3 konform ist zu SQL2 an [DD97]. Sind einige Konzepte gar nicht oder nur in modifizierter Form in kommerziellen DB-Systeme (insbesondere DB/2 V.7.x von IBM, O8i von Oracle) implementiert worden, so wird explizit darauf hingewiesen.

Vorgehensweise:

Damit SQL und Datalog auf einer gemeinsamen sprachlichen Grundlage erörtert werden können, wird für die Definition von SQL das gleiche Vokabular wie bei logikbasierten Sprachen verwendet. In den bisherigen Publikationen werden SQL-Ausdrücke immer

[13] Zum SQL3-Standard gehören fünf Veröffentlichungen: 1. Framework, 2. Foundation, 3. Call-Level Interface (CLI), 4. Persistent Stored Modules (PSM), 5. Host Language Bindings (Bindings).

in BNF-Syntaxdiagrammen zusammen mit informellen Erläuterungen der Funktionalität und Restriktionen spezifiziert. In Fußnoten werden hier die für SQL-Entwickler bekannteren Bezeichner, wie sie bei ANSI ([ANSI99I], [ANSI99II]) und Date/Darwen ([DD97]) verwendet werden, aufgeführt.

Um die nachfolgenden Themen anschaulich diskutieren zu können, wird SQL3 des ANSI-Standards in zwei Schritten normalisiert.

1. **SQL3-Basisfunktionalität:**
   Im ersten Schritt wird eine SQL3-Basisfunktionalität definiert, indem im wesentlichen auf Operatoren mit sehr spezieller Funktionalität verzichtet wird wie z.B. auf Aggregatfunktionen und einige Vergleichsoperatoren. Diese Reduktionen zu einer Art Basis-SQL bedeuten nicht nur eine Einschränkung der Syntax sondern teilweise auch eine Beschränkung der Semantik. Ziel ist es, SQL3 auf eine für die Diskussion inkrementeller Verfahren im SQL-Kontext (Kapitel 5) relevante und mit Datalog vergleichbare Funktionalität zu beschränken. Trotz der Unterschiede zum SQL3-Standard werden die DB-Objekte im weiteren als SQL3- bzw. SQL-Objekte bezeichnet.

2. **Normalisiertes SQL3 (NSQL):**
   Ohne die Semantik weiter einzuschränken wird die Syntax dieses Basis-SQL3 auf diejenigen SQL-Operatoren reduziert, für die es Datalog-Operatoren mit entsprechender Funktionalität gibt. Auf diese Weise werden die Analogien zwischen beiden Sprachen offensichtlicher und die in Abschnitt 3.2 definierten Transformationsschritte intuitiv verständlich. NSQL wird in Abschnitt 3.1 definiert.

Die in den Abschnitten 2.2.1 - 2.2.3 für die Bereitstellung der **Basisfunktionalität** definierten SQL3-Anfragen, Sichten und Integritätsbedingungen sind hinsichtlich der folgenden Eigenschaften gegenüber dem SQL3-Standard eingeschränkt:

1. keine Duplikate,

2. keine Aggregatfunktionen und keine Gruppierungsklauseln,

3. keine SQL- und benutzerdefinierten Funktionen,

4. keine 'Sondervergleichsoperatoren' außer dem EXISTS-Operator,

5. keine NULL-Zustände von Attributen und
   keine UNKNOWN-Ergebnisse von Bedingungen.

Wie bereits zu Beginn von Kapitel 2 bemerkt, basiert SQL auf dem Verständnis, daß eine Relation eine Multimenge ist. Codd ebenso wie Date weisen in [Codd88] und [DD97] 8.3 auf die Problematik hin, daß SQL in diesem Punkt vom relationalen Modell abweicht. In der Praxis werden Schlüssel definiert, um die Eindeutigkeit der gespeicherten Tupel zu garantieren, analog zu der i.d.R. gegebenen Eindeutigkeit der Objekte in der realen Welt. Somit bedeutet die erste Restriktion keine Beschränkung mit wesentlicher praktischer Relevanz und es kann in den Ausdrücken auf die SQL-Option '[ALL|DISTINCT]' (mit/ohne Duplikate) verzichtet werden.

Da Restriktion 2 Aggregatfunktionen (COUNT, SUM, MIN, MAX, AVG, VAR,...) wie auch Gruppierungsklauseln (GROUP BY, HAVING) ausschließt, sind kumulierende Anfragen im eingeschränkten SQL3 nicht formulierbar. Die Restriktionen 2, 3 und 4 stellen semantische Einschränkungen dar. Da diese Konzepte jedoch nicht zu den Basiskonzepten von Datalog zählen, sondern als Erweiterungen diskutiert werden ([AH88],[Ull89],[CGH94]), und es ein Ziel dieser Arbeit ist, die grundsätzlichen Probleme bei der Übertragung inkrementeller Verfahren aus dem Datalog-Kontext nach SQL zu diskutieren, bleiben solche Spezialeigenschaften unberücksichtigt. SQL verfügt über eine Vielzahl von Vergleichsoperatoren[14] mit einer zum Teil sehr speziellen Semantik (BETWEEN, IN, LIKE, IS, ANY, SOME, ALL, EXISTS, UNIQUE, MATCH, OVERLAPS, SIMILAR, DISTINCT, TYPE). Date definiert die Semantik vieler dieser Operatoren auf der Basis arithmetischer Vergleichsoperatoren ([DD97] 12). Viele Sonderoperatoren[15] sind zudem redundant zueinander wie z.B. =ANY, IN, EXIST oder ≠ALL, NOT IN, NOT EXIST mit denen (Nicht-)Elementbeziehungen formuliert werden können. Operatoren, wie z.B. LIKE, UNIQUE, MATCH, SIMILAR, TYPE, DISTINCT, verfügen über eine so spezielle Semantik, daß Datalog um entsprechende Funktionen erweitert werden müßte. Von diesen speziellen Operatoren wird im weiteren nur der EXISTS-Operator verwendet.

Da die Berücksichtigung von unbekannten Attributwerten (NULL/UNKNOWN)[16] nur bei der Auswertung von Bedingungen und der Interpretation ihrer Ergebnisse relevant ist, diese aber keinen Einfluß auf Verfahren der Änderungspropagierung nehmen, werden sie hier nicht weiter berücksichtigt (Restriktion 5).

Die in Abschnitt 2.2.4 definierte SQL-Änderungssprache ist nur insofern gegenüber der des SQL-Standards eingeschränkt, als daß Basis-SQL-Formeln in den bedingten Anweisungen zugelassen sind. Das Transaktionskonzept des SQL-Standards hingegen wird unverändert übernommen, ebenso wie das SQL-Triggerkonzept in Abschnitt 2.2.5. Für die Anfragen, Sichten und Integritätsbedingungen wird in den nachfolgenden Abschnitten nur die Syntax nicht aber die Semantik definiert. Die Semantik dieser SQL-Ausdrücke ergibt durch die in Abschnitt 3.2.2 formulierten Transformationen in semantisch äquivalente Datalog-Ausdrücke, deren Semantik in Abschnitt 2.1.2 definiert ist. Daß den Transformationsvorschriften NSQL zugrunde liegt, stellt keine Einschränkung dar, da NSQL gegenüber Basis-SQL nur syntaktisch normalisiert ist. Auf die Redundanzen zwischen den verschiedenen SQL-Ausdrücken wird explizit hingewiesen. Beispiele für SQL-Sichten und ASSERTIONs werden auch erst für NSQL in Abschnitt 3.1 modelliert.

2.2.1 Anfragen

Anders als beim bereichskalkülbasierten Datalog, wo Variablen die Werte einzelner Attribute einer Relation (Bereichsvariablen) repräsentieren, stehen Variablen[17] im tupelkalkülbasierten SQL für ganze Tupel. Eine **SQL-Tupelvariable** ist eine alphanumeri-

[14] [ANSI99II] 8: <predicate>, [DD97] 12: simple condition

[15] Bzgl. NULL-wertiger Operanden führen diese Operatoren teilweise zu unterschiedlichen Ergebnissen. Eine sehr fundierte Diskussion dieser Problematik führt Date in [DD97] 12, 16.

[16] Das SQL-NULL-Konzept wird in [Klein97], [KK93], [Date86], [Codd86], [Codd87], [Codd88] sehr kritisch diskutiert. Date gibt in [DD97] 16.6 die Empfehlung 'Avoid nulls.'.

[17] [ANSI99II] 7.6: <correlation name>, [DD97] 11: range-variable; In SQL werden entweder die Relationssymbole selbst oder Aliasnamen für Relationen als Tupelvariablen verwendet.

sche Zeichenkette mit bis zu 128 Zeichen. Als Sonderzeichen ist '_' und eine Indizierung[18] der Namen zugelassen. Ein **SQL-Term** ist eine Konstante a,b,c,..,1,2,3,... oder ein **Attributterm** $T_R.A$, wobei T_R eine Tupelvariable bezeichnet und A ein Attribut der Relation, an die T_R gebunden ist.

Anders als in Datalog, wo zwei Arten atomarer Formeln (DB- und Vergleichsatome) bekannt sind, beschränkt sich der Begriff der atomaren Formel in SQL auf Vergleichsatome. Die Typisierung der Tupelvariablen erfolgt in der FROM-Klausel von SELECT-Ausdrücken bzw. in Join-Ausdrücken.

Definition 2.2.1 (Atomare SQL-Formel)
Eine atomare SQL-Formel (SQL-Atom) ist ein Ausdruck der Form

$t_i \ \Theta \ t_j$,

wobei t_i, t_j Terme sind und Θ eines der Vergleichssymbole '$=, >, <, \geq, \leq, \neq$[19] '.

Definition 2.2.2 (SQL-Formel)

1. *Die aussagelogischen Konstanten TRUE und FALSE[20] sind SQL-Formeln[21].*
2. *Jede atomare SQL-Formel ist eine SQL-Formel.*
3. *Sind F, F_1, F_2 SQL-Formeln, dann ist jeder Ausdruck der Form*

   $(\ F_1\ AND\ F_2\ )$ $(\ F_1\ OR\ F_2\ )$ $NOT\ (\ F\ )$

   eine SQL-Formel.
4. *Ist Q eine SQL-Anfrage[22], dann ist jede existenzquantifizierte SQL-Anfrage*

   $EXISTS\ (\ Q\ )$ $NOT\ EXISTS\ (\ Q\ )$

   eine SQL-Formel.

Für die Darstellung eines Fakts fehlen im SQL-Standard die Vorgaben. Da auch in SQL verschiedentlich positionelle Schreibweisen verwendet werden, wird im weiteren die aus Datalog bekannte Notation auch für SQL-Fakten verwendet.

---

[18] Eine Indizierung ist im SQL-Standard nicht zulässig. Für die Symbole von Tupelvariablen gelten in SQL die gleichen Restriktionen wie für die Bezeichner von Schemaobjekten der Relationen, Attributen, Assertions,... ([ANSI99II] 4.2, 5.2, [DD97] 3.4)

[19] Ungleichheit wird in SQL nicht durch '≠' ausgedrückt, sondern durch '<>' oder '!='. Der Einfachheit halber wird das bereits in Datalog eingeführte und in der Mathematik weit verbreitete Symbol aber beibehalten.

[20] Da NULL-wertige Operanden in SQL zulässig sind, ist als dritter Wahrheitswert UNKNOWN definiert. Da NULL-Werte in dieser Arbeit nicht weiter berücksichtigt werden, wird auf die Einführung einer dreiwertigen Logik verzichtet.

[21] [ANSI99II] 8: <search condition>, [DD97] 12: conditional expression

[22] [ANSI99II] 7.12: <query epression>, [DD97] 11: table expression

Definition 2.2.3 (SQL-Anfrage)

1. *Jeder SF-Ausdruck*[23] *der Form*

   $$\textit{SELECT}\ \ t_1,\ \ldots,\ t_n\quad \textit{FROM}\ \ R_1\ \textit{AS}\ \ T_1,\ \ldots,\ R_m\ \textit{AS}\ \ T_m$$

   ist eine SQL-Anfrage,
   wobei $t_1, \ldots, t_n$ SQL-Terme sind ($n \geq 1$). $R_1, \ldots, R_m$ sind die Symbole von Basisrelationen, Sichten, Hilfssichten und $T_1, \ldots, T_m$ sind die ihnen zugeordneten SQL-Tupelvariablen[24] *($m \geq 1$). Als Tupelvariablen werden in den Termen $t_1, \ldots, t_n$ ausschließlich die Tupelvariablen $T_1, \ldots, T_m$ verwendet.*

2. *Jeder SFW-Ausdruck*[25] *der Form*

   $$SF\quad \textit{WHERE}\ \ F$$

   ist eine SQL-Anfrage,
   wobei SF ein SF-Ausdruck und F eine SQL-Formel ist. In F treten nur Terme auf, deren Tupelvariablen in der FROM-Liste des SF-Ausdrucks gebunden sind oder in einer übergeordneten Anfrage, falls der SFW-Ausdruck eine Unteranfrage ist (correlated subquery).

3. *Jeder JO-Ausdruck*[26] *der Form*

   $$R_1\quad join\_operator\quad R_2$$

   $$R_1\ \textit{AS}\ \ T_1\ \ join\_type\ \ \textit{JOIN}\ \ R_2\ \textit{AS}\ \ T_2\ \ \textit{ON}\ F$$

   ist eine SQL-Anfrage,
   wobei R_1 und R_2 Relationssymbole sind oder geklammerte Anfrageausdrücke der Form (Q). Zulässige Join-Operatoren sind CROSS JOIN, NATURAL join_type JOIN, UNION JOIN. 'join_type' bezeichnet die Art der Join-Verknüpfung näher: INNER, OUTER LEFT, OUTER RIGHT, OUTER FULL. F ist eine SQL-Formel, in der als Tupelvariablen nur die beiden Tupelvariablen T_1 und T_2 auftreten.

4. *Jeder SO-Ausdruck*[27] *der Form*

   $$Q_1\quad set\_operator\quad Q_2$$

[23] [ANSI99II] 7.11: <query specification>, [DD97] 11.6: select expression. Die beiden in SQL optionalen GROUP BY-, HAVING-Klauseln, die i.d.R. zusammen mit Aggregatfunktionen verwendet werden, sind nicht Gegenstand dieser Arbeit.

[24] In SQL ist die Verwendung von Tupelvariablen (Aliasnamen) optional. Um hervorzuheben, daß die SELECT-Ausdrücke auf dem TRC basieren, ist ihre Verwendung im weiteren zwingend.

[25] [ANSI99II] 7.11: <query specification>, [DD97] 11.6: select expression.

[26] [ANSI99II] 7.7: <joined table>, [DD97] 11.2: join-table-expression

[27] [ANSI99II] 7.12: <non-join query expression>, [DD97] 11.4: nonjoin table expression.

ist eine SQL-Anfrage,
wobei Q_1 und Q_2 SQL-Anfragen sind, die nicht vom Typ einer WITH-Anfrage sind. Die Anzahl der Attribute der Ergebnislisten der beiden Anfragen muß identisch sein (vereinigungskonforme Struktur)[28]*. 'set_operator' bezeichnet einen der drei Mengenoperatoren: UNION (Vereinigung), EXCEPT (Differenz), INTERSECT (Durchschnitt).*

5. *Jeder WITH-Anfrageausdruck*[29] *der Form*

   $$\textit{WITH}\ [\ \textit{RECURSIVE}\ ]\quad S_1, \ldots, S_n \quad Q$$

   ist eine SQL-Anfrage,
   wobei $S_1, \ldots, S_n$ Hilfssichten (Definition 2.2.4) sind und Q eine SQL-Anfrage, die nicht vom Typ eines WITH-Anfrageausdrucks ist ($n \geq 1$). Als Relationssymbole sind in den Anfrageausdrücken der Hilfssichten $S_1, \ldots, S_n$ und in Q neben denen der persistenten Relationen auch die Sichtsymbole von $S_1, \ldots, S_n$ zulässig. Die RECURSIVE-Option ist nur notwendig, wenn die Hilfssichten $S_1, \ldots, S_n$ rekursiv sind.

SF-Ausdrücke werden auch als unbedingte und SFW-Ausdrücke als bedingte Anfragen bezeichnet. Diese beiden Anfragearten werden wiederum als SELECT-Ausdrücke mit optionaler WHERE-Klausel bezeichnet.

Die Struktur der Ergebnismenge einer Anfrage wird je nach Anfragetyp unterschiedlich definiert. Bei SF/SFW-Ausdrücken wird sie durch die SQL-Terme der SELECT-Liste $(t_1, \ldots, t_n)$ bestimmt. Bei einem SO-Ausdruck ist der erste Operand bestimmend hinsichtlich der Attributsymbole der Ergebnisliste. Die Ergebnismengen beider Operanden sind vereinigungskonform. Handelt es sich um ein Relationssymbol, so ist die Attributliste der Relation auch die Ergebnisliste des Operanden. Die Ergebnisliste eines JO-Ausdrucks wird durch die Verkettung der Ergebnislisten beider Operanden in Analogie zum Kreuzprodukt in der RA gebildet. Die Ergebnisliste einer WITH-Anfrage ist die Ergebnisliste der Anfrage Q.

Aufgrund der syntaktischen Restriktionen, daß das Ergebnistupel einer Anfrage bzw. Sicht immer über mindestens ein Attribut verfügen muß, und daß EXISTS(Q) nur eine Formel, nicht aber eine Anfrage ist, sind parameterlose Anfragen in SQL nicht formulierbar. Sie können nur mittels existenzquantifizierter Unteranfragen in einer Formel simuliert werden. Da parameterlose Anfragen keine zusätzlichen Schwierigkeiten für Änderungspropagierungsverfahren bedeuten, werden sie auch nicht in NSQL eingeführt.

Syntaktisch sind zwei Arten von SQL-Anfragen zu unterscheiden, die TRC-basierten SELECT-Ausdrücke und die RA-basierten SO-/JO-Ausdrücke. Die Abbildung der JOIN-Operatoren auf die aus der RA bekannten Join-Operatoren kann unmittelbar erfolgen und ist trivial. Date zeigt in [DD97] 11.2 wie die JOIN-Operatoren durch SELECT-Ausdrücke formuliert werden können, indem er die SELECT-Ausdrücke zur Semantik-Definition

[28] Da in SQL für jedes Attribut der Datentyp und die Attributlänge spezifiziert wird (Sortenrestriktion), beinhaltet der Begriff der 'Vereinigungskonformität' zudem die Forderung der kompatiblen Datentypen und Datenlängen der Attribute.

[29] [ANSI99II] 7.12: <query expression>, [DD97] 11: table expression. Da sich [DD97] noch auf SQL2 bezieht, fehlt die WITH-Klausel und die Möglichkeit, rekursive (Hilfs-)Sichten und Anfragen zu definieren. Der SQL2-table expression entspricht dem SQL3-query expression body.

der JO-Ausdrücke verwendet. Aufgrund ihrer semantischen Redundanz zu SELECT-Ausdrücken werden JO-Ausdrücke nicht weiter betrachtet. NSQL beschränkt sich syntaktisch auf Anfragen in Form von SELECT- und SO-Ausdrücken. Welche Bedeutung DBS-Hersteller dieser syntaktisch vereinfachten Schreibweise der JO-Ausdrücke beimessen, zeigt sich u.a. beim Oracle DB-System, wo diese Anfrageteilausdrücke nicht implementiert worden sind. Bei der DB2 sind sie nur als geschachtelte Unteranfragen in der FROM-Liste eines SELECT-Ausdrucks zugelassen.

2.2.2 Sichten

Sichten in SQL sind immer virtuelle Sichten, da dem Standard ein Konzept materialisierter Sichten fehlt.[30]

Definition 2.2.4 (SQL-Sicht)
Jeder Ausdruck der Form

$$S\ (\ A_1, \ldots, A_n\ )\ \ AS\ \ Q$$

ist eine SQL-Sicht,
wobei S Sichtsymbol ist und $A_1, \ldots, A_n$ Attributsymbole ($n \geq 1$). Q ist eine SQL-Anfrage. Die Anzahl n der Attribute der Sicht muß identisch sein mit der Anzahl der Terme der Ergebnisliste der Anfrage Q.

Die Zuordnung zwischen den Termen der Ergebnisliste der SQL-Anfrage Q und den Attributen der Sicht erfolgt gemäß der in der Sichtdefinition spezifizierten Reihenfolge (positionelle Darstellung). Parameterlose Sichten sind anders als in Datalog nicht formulierbar. Eine solche Sichtspezifikation stellt in SQL keinen eigenständigen Befehl dar. Sie kann zur Spezifikation einer Hilfssicht in der WITH-Klausel eines Anfrageausdrucks verwendet werden. Zudem kann eine virtuelle Sicht mittels des DDL-Befehls CREATE VIEW als persistentes Schemaobjekt in einem DB-System definiert werden.

CREATE [RECURSIVE] VIEW $S\ (\ A_1, \ldots, A_n\ )$ AS Q

Die RECURSIVE-Option muß nur für rekursive Sichten spezifiziert werden. Rekursive Sichten[31] können in SQL sowohl als persistente wie auch als Hilfssichten definiert werden. Die Verankerung der Rekursion muß mittels UNION-Operator in mindestens einer der beteiligten Sichten definiert werden. Mengenoperatoren, die nicht der Verankerung dienen, sind nicht zugelassen.

Beispiel 2.2.5 (Rekursion)
Spezifikation der transitiven Hülle p einer Relation r:

- *Datalog:* $p(X,Y) \leftarrow r(X,Y)$ – *Verankerung*
  $p(X,Y) \leftarrow r(X,Z) \wedge p(Z,Y)$ – *lineare Rekursion*

[30] In Abschnitt 5.4 wird das SQL-Sichtenkonzept um materialisierte Sichten ergänzt.

[31] [ANSI99II] 7.12: potentially recursive <with clause>, [DD97] F.3: recursive table expression, [GP99] 32: recursive union, wobei lt. Date und Celko dieser Begriff ein vom Standardisierungsgremium verworfenes Rekursionskonzept bezeichnet.

- *SQL mit rekursiver Sicht p:*

$$
\begin{array}{lll}
p(a_1, a_2\ )\ AS\ SELECT\ t_r.a_1,\ t_r.a_2\ FROM\ r\ AS\ t_r & & - \textit{Verankerung} \\
\quad UNION & & \\
\quad SELECT\ t_r.a_1,\ t_p.a_2 & & - \textit{lineare Rekursion} \\
\quad FROM\ \ r\ AS\ t_r,\ \ p\ AS\ t_p & & \\
\quad WHERE\ t_p.a_2\ =\ t_r.a_1 & &
\end{array}
$$

- *SQL mit rekursiven Hilfssichten:*

$$
\begin{array}{l}
p(a_1, a_2)\ AS\ WITH\ RECURSIVE \\
\quad h(a_1, a_2)\ \ AS\ \ SELECT\ t_r.a_1,\ t_r.a_2\ FROM\ r\ AS\ t_r \\
\qquad\qquad\quad UNION \\
\qquad\qquad\quad SELECT\ t_r.a_1,\ t_h.a_2 \\
\qquad\qquad\quad FROM\ \ r\ AS\ t_r,\ \ h\ AS\ t_h \\
\qquad\qquad\quad WHERE\ t_h.a_2\ =\ t_r.a_1 \\
\quad SELECT\ t_h.a_1,\ t_h.a_2\ \ FROM\ \ h\ AS\ t_h.
\end{array}
$$

In SQL sind ausschließlich linear rekursive[32] Sichten zugelassen.

1. Seien S_i, S_j, S_k von einander rekursiv abhängige Sichten. S_i, S_j, S_k sind linear rekursiv, gdw. jede der rekursiv abhängigen Sichten in ihrem Anfrageausdruck jeweils nur von höchstens einer anderen Sicht rekursiv abhängt.

2. Seien S_i, S_j, S_k von einander rekursiv abhängige Sichten und treten S_j und S_k in der Spezifikation von S_i auf. S_i, S_j, S_k sind linear rekursiv, gdw. S_j und S_k weder in der FROM-Klausel noch gemeinsam in einer Unteranfrage auftreten (nur in 'unabhängigen' Unteranfragen).

Aufgrund der Restriktion der linearen Rekursion können nachfolgende nicht lineare Datalog-Regeln nicht als SQL-Sicht formuliert werden.

Beispiel 2.2.6 (Nichtlineare Rekursion)
Spezifikation der transitiven Hülle p einer Relation r:

- *Datalog:*

$$
\begin{array}{lll}
p(X,Y)\ \leftarrow\ r(X,Y) & & - \textit{Verankerung} \\
p(X,Y)\ \leftarrow\ p(X,Z)\ \wedge\ p(Z,Y) & & - \textit{nicht lineare Rekursion.}
\end{array}
$$

Der in Datalog verwendete Begriff der linearen Rekursion ist restriktiver und schließt das mehrfache Auftreten rekursiv abhängiger Regeln im Regelrumpf grundsätzlich aus (entspricht in SQL Punkt 1). Im Datalog-Kontext sind für linear rekursive Regelmengen eine Reihe von Optimierungen bei verschiedenen Problemstellungen entwickelt worden (u.a. semi-naive Auswertung [CGT90] 8.4). Der Unterschied in beiden Definitionen ist so gravierend, daß analysiert werden muß, ob die Optimierungsstrategien aus dem Datalog-Kontext korrekt in SQL anwendbar sind.

[32] [ANSI99II] 7.12 Punkt 1)g). Zudem ist lineare Rekursion eine der Einschränkungen hinsichtlich der Verwendbarkeit der <search clause> (Punkt 1)g + h)), die hier nicht weiter erörtert wird.

IBM hat bislang als einziger DBS-Hersteller diese WITH-Option und damit das Konzept der (rekursiven) Hilfssichten implementiert. Groff/Weinberg erläutern dieses Konzept in [GW99] detaillierter.

Die für Datalog in Abschnitt 2.1.1 eingeführten Definitionen der Mengen von Basisrelations- und Sichtsymbolen $rel(R)$, $rel_{body}(R)$, $rel_{head}(R)$, $rel_{edb}(\mathcal{R})$, $rel_{idb}(\mathcal{R})$, $rel_{virt}(\mathcal{R})$, $rel_{mat}(\mathcal{R})$, $rel_{help}(\mathcal{R})$ gelten für SQL analog. R repräsentiert eine SQL-Sichtdefinition oder ASSERTION und $\mathcal{R}$ eine Menge der Sichtdefinitionen und ASSERTIONs. Für die Menge der Symbole materialisierter Sichten gilt, daß $rel_{mat}(\mathcal{R})= \emptyset$.

Im Datalog-Kontext wird eine Diskussion über die Bereichsbeschränkung von Regeln und Anfragen (Definitionen 2.1.5, 2.1.8) geführt. In SQL ist diese Problematik für Anwender nicht ersichtlich. In den Kapiteln 7, 8, 11.21 in [ANSI99II] werden die erforderlichen syntaktischen Restriktionen für Anfragen, Bedingungen und Sichtdefinitionen formuliert, die sicherstellen, daß keine unsicheren SQL-Anfragen/Sichten syntaktisch gültige SQL-Ausdrücke sind. In diesem Zusammenhang wird der Begriff der Sicherheit bzw. Bereichsbeschränkung nicht erwähnt, noch werden Gründe für diese Einschränkungen genannt. Die nachfolgend aufgeführten syntaktischen Restriktionen für eine gültige SQL-Anfrage gewährleisten die Bereichsbeschränkung dieser Anfrage.

1. Alle in einer Anfrage verwendeten Tupelvariablen müssen in der FROM-Liste an eine Relation gebunden sein (gültige Tupelvariable).
2. Alle in einer Anfrage verwendeten Terme sind entweder Konstanten oder Attributterme mit einer gültigen Tupelvariablen.
3. In der FROM-Liste kann Negation nicht auftreten.

Die Punkte 1 und 2 würden in Datalog gewährleisten, daß keine Bereichsvariablen ohne Bindung in DB-Atomen auftreten, und Punkt 3, daß die Bindung mit mindestens einem positiven DB-Atomen erfolgt. Liegt einer Sicht eine bereichsbeschränkte Anfrage zugrunde, so ist die Sicht auch bereichsbeschränkt. Eine syntaktisch korrekte SQL-Anfrage/Sicht kann nur in eine sichere Datalog-Anfrage/Sicht transformiert werden (Abschnitt 3.2).

Aus Gründen der Vollständigkeit sei auf zwei neue Anfrage-Klauseln in SQL3 hingewiesen, die jedoch über keine Relevanz für das Thema dieser Arbeit verfügen. Mit der <search clause> kann die Suchreihenfolge (Traversierung) für die Bearbeitung rekursiver Anfragen spezifiziert werden: Tiefensuche (depth first) oder Breitensuche (breadth first). Mit der <cycle clause> können für den Fall, daß Duplikate auftreten, Terminierungsereignisse für die Bearbeitung rekursiver Anfragen formuliert werden.

2.2.3 Integritätsbedingungen

Ausschließlich statische Integritätsbedingungen sind definierbar. In den Formeln der Bedingungen fehlen die syntaktischen Konstrukte[33], um auf den Zustand vor und nach einem Manipulationsereignis zugreifen zu können. Werden SQL-Integritätsbedingungen gemäß der Art ihrer Definition klassifiziert, so werden vier Klassen unterschieden.

[33]Solche Referenzvariablen/-relationen stehen seit SQL3 im Bedingungs- und Aktionsteil von Triggern zur Verfügung. Zur Formulierung dynamischer Integritätsbedingungen wird daher auf das aktive Regelkonzept (Trigger) von SQL3 verwiesen (Abschnitt 2.2.5).

1. Domänen-Constraint ('domain constraint'): CHECK-Bedingung
2. Attribut-Constraint ('column constraint'): NOT NULL[34], PRIMARY KEY, UNIQUE KEY, FOREIGN KEY, CHECK-Bedingung
3. Basisrelationen-Constraint ('table constraint'): PRIMARY KEY, UNIQUE KEY, FOREIGN KEY, CHECK-Bedingung
4. ASSERTION: CHECK-Bedingung

Domänen- und Attribut-Constraints sind Integritätsbedingungen, die den Wert nur eines Attributs beschränken. Da beide Constraint-Arten abkürzende Notationen für Basisrelationen–Constraints darstellen, wird auf ihre Definition verzichtet. Der Nachweis, daß jedes Domänen- als Attribut-Constraint sowie jedes Attribut- als Basisrelationen–Constraint und letztlich jedes Basisrelationen–Constraint (ohne Fehlerkorrektur) als ASSERTION formulierbar ist, wird bereits im Dokument des SQL-Standards[35] geführt. Dort wird die Semantik dieser Constraints auf die Semantik von ASSERTIONs zurückgeführt. Somit bedeutet die Beschränkung der Basis-SQL- und NSQL-Integritätsbedingungen (Abschnitt 3.1) keinen Verlust an Ausdruckskraft. Obwohl auch Basisrelationen-Constraints semantisch redundant zu ASSERTIONs sind, werden sie in diesem Abschnitt trotzdem diskutiert, um einigen Unklarheiten und Mißverständnissen aus der Sekundärliteratur zu begegnen. Sie werden jedoch nur informell eingeführt. Die inkrementellen Integritätsprüfungsverfahren in Kapitel 5 werden ausschließlich für ASSERTIONs diskutiert.

ASSERTIONs sind die einzigen SQL-Integritätsbedingungen, die als eigenständige DB-Objekte erzeugt werden (CREATE ASSERTION). Als Bedingung einer ASSERTION ist eine SQL-Formel zugelassen. Wenig intuitiv ist die Formulierung von Allaussagen als ASSERTIONs, da in SQL kein Allquantor-Symbol[36] zur Verfügung steht, und eine Allaussage explizit mittels eines negiertem EXISTS-Operators formuliert werden muß.

Definition 2.2.7 (SQL-ASSERTION)
Jeder Ausdruck der Form

IC CHECK (F) *check_time_clause*

ist eine SQL-ASSERTION[37]*,*
wobei IC ein ASSERTION-Symbol und F eine SQL-Formel ist. check_time_clause spezifiziert den Prüfungszeitpunkt:

INITIALLY { *DEFERRED* | *IMMEDIATE* }.

[35] ANSI-Standard: [ANSI99II] 11; Date: [DD97] 14; Melton/Simon: [MS93] 10.3, S. 211, [BMP01]

[36] In der Literatur wird teilweise der Vergleichsoperator '=ALL' als Allquantor bezeichnet. Mit diesem Operator kann zwar eine Allaussage formuliert werden, es handelt sich dabei aber um einen binären Operator. Die aus der Prädikatenlogik bekannten Formulierungen mit unären Quantoren lassen sich nur schwer lesbar in binäre Vergleiche transformieren.

[37] In der Literatur werden die Begriffe Integritätsbedingung, Constraint und Assertion vielfach synonym verwendet. Im SQL-Kontext wird mit dem Begriff ASSERTION aber nur diese Form einer Integritätsbedingung bezeichnet.

Anders als im Datalog-Kontext sind in SQL zwei Integritätsprüfungszeitpunkte zulässig, IMMEDIATE und DEFERRED. Mit IMMEDIATE wird der Zeitpunkt unmittelbar nach der Ausführung einer Änderungsanweisung bezeichnet und mit DEFERRED der Zeitpunkt zum Transaktionsende (COMMIT-Anweisung). Zum IMMEDIATE-Prüfungszeitpunkt sind bereits alle von der aktuellen Änderungsanweisung betroffenen Basisfakten manipuliert und die relevanten Integritätsbedingungen werden reihenfolgeunabhängig geprüft. Der während des Schemaentwurfs definierte Prüfungszeitpunkt kann während einer Transaktion umgesetzt werden, wenn dies nicht explizit bei der Definition ausgeschlossen wurde. Im weiteren wird jedoch vorausgesetzt, daß der zur Schemaentwurfszeit spezifizierte Prüfungszeitpunkt unveränderlich ist.

Da eine SQL-Transaktion eine Folge von Änderungsanweisungen ist, wird bei einem Integritätsfehler jeweils zum letzten konsistenten (Zwischen-)Zustand zurückgerollt. Im Rahmen einer IMMEDIATE-Prüfung wird mit dem Zurückrollen nur der fehlerverursachenden Änderungsanweisung reagiert (lokales ROLLBACK) und im Rahmen einer DEFERRED-Prüfung mit dem Zurückrollen der gesamten Transaktion ((globales) ROLLBACK). Der Vorgang des Zurückrollens schließt neben allen direkt durch Anwender manipulierten Fakten auch die durch FOREIGN KEY-Korrekturen oder durch Trigger manipulierten Fakten mit ein. Im SQL-Kontext gilt die Eigenschaft der atomaren Ausführung nicht nur für Transaktionen mit ihren DEFERRED-Prüfungen sondern auch für eine einzelne Änderungsanweisung mit ihren IMMEDIATE-Prüfungen.

Drei vordefinierte Schlüsselbedingungen und das sogenannte CHECK-Constraint werden unterschieden.

1. UNIQUE KEY: CONSTRAINT IC UNIQUE ($A_1, \ldots, A_n$)

   Die Attribute $A_1, \ldots, A_n$ ($n \geq 1$) bilden einen Eindeutigkeitsschlüssel.

2. PRIMARY KEY: CONSTRAINT IC PRIMARY KEY ($A_1, \ldots, A_n$)

   Die Attribute $A_1, \ldots, A_n$ ($n \geq 1$) bilden den Primärschlüssel[38] der Basisrelation.

3. FOREIGN KEY: CONSTRAINT IC FOREIGN KEY ($A_1, \ldots, A_n$)
   REFERENCES R($A_1^R, \ldots, A_n^R$)
   [ON UPDATE referential_action]
   [ON DELETE referential_ action]

   Zwischen den Attributen $A_1, \ldots, A_n$ der zugehörigen Basisrelation und den Attributen $A_1^R, \ldots, A_n^R$ der referenzierten Basisrelation R wird eine Fremdschlüsselbeziehung definiert. Die ON UPDATE-/ON DELETE-Klauseln sind optional und spezifizieren die Reaktionen des DB-Systems auf einen Integritätsfehler. Zugelassene Reaktionen (referential_action) sind: CASCADE, SET NULL, SET DEFAULT, RESTRICT, NO ACTION.

4. CHECK: CONSTRAINT IC CHECK (F)

   F ist eine SQL-Formel, in der die Attributsymbole der zugehörigen Basisrelation als Terme ohne Tupelvariable auftreten.

[38] Für PRIMARY KEY-Attribute sind keine NULL-Werte zulässig (implizite NOT NULL-Bedingung).

Im Gegensatz zu ASSERTIONs sind Basisrelationen-Constraints keine eigenständigen DB-Objekte und werden im Rahmen eines CREATE TABLE-Befehls mit der CONSTRAINT-Klausel spezifiziert. Ihre Existenz ist unmittelbar mit der zugehörigen Basisrelation verknüpft. Für eine Basisrelation sind mit Ausnahme des PRIMARY KEYs, der maximal einmal definiert werden kann, von jeder Constraint-Art beliebig viele Integritätsbedingungen definierbar. Alle Constraint-Klauseln sind optional um die Angabe des Prüfungszeitpunkts (*check_time_clause*) erweiterbar. FOREIGN KEY-Constraints erweitern das Konzept der Integritätsprüfung ('integrity checking') um die Möglichkeit zur Fehlerkorrektur ('integrity repair'). Mittels der CASCADE-Option werden bei Löschungen und Modifikationen von Tupeln der referenzierten Basisrelation alle abhängigen Tupel gelöscht. Ist SET NULL / SET DEFAULT spezifiziert, dann werden für alle abhängigen Tupel die Werte der Attribute $A_1, ..., A_n$ durch NULL oder durch den ihnen zugeordneten DEFAULT-Wert ersetzt. Bei der RESTRICT / NO ACTION-Option werden keine fehlerkorrigierenden Aktionen sondern ein ROLLBACK ausgeführt.

Beispiel 2.2.8 (SQL-Integritätsbedingungen)
Seien die Basisrelationen p mit zwei Attributen und q mit einem Attribut gegeben. In den verschiedenen DB-Sprachen wird eine Fremdschlüsselbeziehung wie folgt definiert:

- *Prädikatenlogik:* $\forall X, Y : \; p(X, Y) \;\Rightarrow\; q(Y)$
- *Datalog:* $ic \;\leftarrow\; \neg\, h; \qquad h \;\leftarrow\; p(X, Y) \wedge \;\neg\, q(Y);$
- *FK für p:* $\mathit{FOREIGN\ KEY}\ (a_2)\ \ \mathit{REFERENCES}\ \ q(a_1)$
- *CHECK für p:*
$$\begin{array}{l}\mathit{CHECK}\ (\ \mathit{EXISTS}\ (\ \mathit{SELECT}\ \ t_q.a_1\ \ \mathit{FROM}\ q\ \ \mathit{AS}\ \ t_q \\ \qquad\qquad\qquad\qquad\qquad \mathit{WHERE}\ \ t_q.a_1 = a_2))\end{array}$$
- *ASSERTION:*
$$\begin{array}{l}\mathit{CHECK}\ (\ \mathit{NOT\ EXISTS}\ \ (\\ \qquad \mathit{SELECT}\ \ t_p.a_1, t_p.a_2\ \ \mathit{FROM}\ \ p\ \ \mathit{AS}\ \ t_p \\ \qquad \mathit{WHERE}\ \ \mathit{NOT\ EXISTS}\ (\ \mathit{SELECT}\ \ t_q.a_1 \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{FROM}\ \ \ q\ \ \mathit{AS}\ \ t_q \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{WHERE}\ \ t_q.a_1 = t_p.a_2))).\end{array}$$

Die Formeln von CHECK-Bedingungen, definiert als ASSERTIONs oder als Basisrelationen-Constraints (CHECK-Constraints), unterscheiden sich syntaktisch deutlich. Die Formel F einer ASSERTION ist eine gültige SQL-Formel (Definition 2.2.2) in der jeder Attributterm über eine Tupelvariable verfügt, die an eine Relation gebunden ist. Die Formel F eines CHECK-Constraints wird als implizit allquantifiziert angenommen und die Attribute der zugehörigen Basisrelation treten in F ohne explizite Quantifizierung und ohne Bindung an eine Tupelvariable auf. Aufgrund der impliziten Allquantifizierung stellen sie Kurzschreibweisen von ASSERTIONs mit führendem NOT EXISTS-Operator dar.

Die semantischen Unterschiede zwischen ASSERTIONs und CHECK-Constraints sind hingegen nicht so offensichtlich. Bei genauerer Betrachtung wird jedoch deutlich, daß zwar jedes CHECK-Constraint als ASSERTION ausgedrückt werden kann, dies aber im umgekehrten Fall nur für ASSERTIONs möglich ist, die Allaussagen sind. Infolge der impliziten Allquantifizierung einer CHECK-Constraint-Formel sind Formeln mit einem führenden

Existenzquantor nicht als CHECK-Constraint sondern nur als ASSERTION formulierbar. Die Prüfungen einer existenzquantifizierten ASSERTION und eines entsprechend formulierten CHECK-Constraints liefern unterschiedliche Ergebnisse, wenn die zum Constraint zugehörige Basisrelation leer ist. Aufgrund der impliziten führenden Allquantifizierung ist das Ergebnis der Anwendung des CHECK-Constraints auf einer leeren Menge TRUE und die Anwendung der ASSERTION aufgrund des führenden EXISTS-Operators FALSE. Die Unterschiede und Gemeinsamkeiten zwischen ASSERTIONs und Constraints werden ausführlich von Behrend et.al. in [BMP01] und Date in [DD97] erörtert.

Neben der Fragestellung, wie sich CHECK-Constraints und ASSERTIONs semantisch unterscheiden, werden in der Sekundärliteratur zwei weitere Fragen teilweise nicht oder auch widersprüchlich beantwortet. Diese Problemstellungen werden ausführlich von Behrend et.al. in [BMP01] erörtert und anhand vieler Zitate belegt. Ebenfalls ausführliche und korrekte Antworten gibt Date in [DD97].

- Sind multirelationale CHECK-Formeln, ASSERTIONs zulässig?
  Im Standard ([ANSI99II] 11.9) sind für die Formel F eines CHECK-Constraints oder einer ASSERTION keine Einschränkungen formuliert. Gegenteilige Annahmen[39] erscheinen vor dem Hintergrund der funktional beschränkten Implementierungen in DB-Systemen kommerzieller Hersteller (Oracle, DB2 u.v.m.) plausibel. Mit den dort zulässigen CHECK-Formeln können keine Vergleiche mit Attributwerten durchgeführt werden, die in (anderen) Basisrelationen gespeichert sind.

- Sind SQL-Integritätsbedingungen auf Sichten zulässig?
  Die Formeln von Constraints und ASSERTIONs können sehr wohl Sichten referenzieren. Ein Constraint kann jedoch nur mittels eines CREATE TABLE-Befehls nicht aber mittels eines CREATE VIEW-Befehls erzeugt werden. Somit können die vordefinierten SQL-Schlüsselbedingungen (UNIQUE, PRIMARY, FOREIGN KEY) nicht für Sichten definiert werden[40] und müssen daher sehr aufwendig (und fehleranfällig) mittels ASSERTIONs simuliert werden. Zudem ist die Auszeichnung eines Eindeutigkeitsschlüssels als Primärschlüssel für Sichten nicht möglich, ebensowenig wie die Ausführung von Fehlerkorrekturmaßnahmen bei FOREIGN KEY-Verletzungen.

Dieses sehr funktionale Konzept statischer Integritätsbedingungen in SQL von 1992 ist bislang von führenden Herstellern kommerzieller DB-Systeme (Oracle 8i, IBM DB2 V7.x u.v.m.) nur rudimentär implementiert worden. ASSERTIONs sind nicht realisiert worden. In CHECK-Bedingungen von Constraints sind nur Vergleiche mit konstanten Werten, aber keine SQL-Anfragen zulässig. Lediglich die NOT NULL- und die Schlüsselbedingungen sind ohne Einschränkungen realisiert worden.

In Vorgriff auf die neue, in Abschnitt 2.2.5 vorgestellte SQL3-Funktionalität, der Trigger, kann eine SQL-Datenbank als aktive, deduktive Datenbank definiert werden. Im weiteren wird immer das SQL3-Triggerkonzept für eine SQL-Datenbank vorausgesetzt.

[39] [GP99] 20, S. 412 "A <Table Constraint> defines a rule, that limits the set of values for one or more columns of a Base table"; 20, S. 437 "CREATE ASSERTION defines a new rule that will constrain the set of valid values for one or more Base tables.";
[Cel00] 1, S. 14 "Column Constraints are rules that are attached to a table [Basisrelation]. All rows in the table are validated against them."

[40] [DD97] Appendix D S. 443: "It is strange that integrity constraints can refer to views but not be stadet as a part of a view definition (contrast the situation with respect to base tables)."

In Hinblick auf die Erweiterung des SQL-Sichtkonzepts um materialisierte Sichten in Abschnitt 5.4 basiert die Definition des DB-Schemas $\mathcal{S}^{\mathcal{S}}$ bereits auf einer Menge materialisierter SQL-Sichten $\mathcal{RM}$, die jedoch für das hier diskutierte SQL3-Sichtkonzept eine leere Menge ist.

Definition 2.2.9 (SQL-Datenbankschema)
Ein SQL-Datenbankschema ist ein Quadrupel $\mathcal{S}^{\mathcal{S}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C}, \mathcal{T} \rangle$ *mit*

1. *einer endlichen, ggf. leeren Menge von Basisrelationen* $\mathcal{B}$,
2. *einer endlichen, ggf. leeren Menge von Sichtdefinitionen* $\mathcal{R} := \mathcal{RV} \cup \mathcal{RM} \cup \mathcal{RH}$,
   - $\mathcal{RV}$ *ist die Menge der Definitionen von virtuellen Sichten,*
   - $\mathcal{RM}$ *ist die Menge der Definitionen von materialisierten Sichten (für SQL3 gilt, daß* $\mathcal{RM} := \emptyset$*),*
   - $\mathcal{RH}$ *ist die Menge der Definitionen von Hilfssichten* R^{help} *aller Sichten* $R \in \mathcal{RV}$,
3. *einer endlichen, ggf. leeren Menge* $\mathcal{C}$ *der Definitionen von* ASSERTIONs,
4. *und einer endlichen, ggf. leeren Menge* $\mathcal{T}$ *der Definitionen von SQL3-Triggern.*

Definition 2.2.10 (SQL-Signatur)
Sei $\mathcal{S}^{\mathcal{S}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C}, \mathcal{T} \rangle$ *ein SQL-DB-Schema, dann ist die Signatur des Schemas ein Quadrupel* $\Sigma^S = \langle\ \mathcal{PS}, \mathcal{AS}, \mathcal{CS}, \mathcal{TS}\ \rangle$ *mit*

1. *einer endlichen, ggf. leeren Menge* $\mathcal{PS}$ *mit Relationssymbolen, wobei* $\mathcal{PS} := rel_{edb}(\mathcal{R}) \cup rel_{virt}(\mathcal{R}) \cup rel_{help}(\mathcal{R})$,
2. *einer endlichen, ggf. leeren Menge* $\mathcal{AS}$ *mit Attributsymbolen*
3. *einer endlichen, ggf. leeren Menge* $\mathcal{CS}$ *mit* ASSERTION*-Symbolen*
4. *und einer endlichen, ggf. leeren Menge* $\mathcal{TS}$ *mit Triggersymbolen.*

Definition 2.2.11 (SQL-Datenbank)
Eine SQL-Datenbank ist ein Tupel $\mathcal{D}^{\mathcal{S}} = \langle \mathcal{F}, \mathcal{S}^{\mathcal{S}} \rangle$ *mit*

1. *einer endlichen, ggf. leeren Menge* $\mathcal{F}$ *von Basisfakten*
2. *und einem DB-Schema* $\mathcal{S}^{\mathcal{S}}$.

2.2.4 Änderungen und Transaktionen

In diesem Abschnitt werden die Änderungssprache und das Transaktionskonzept des Basis-SQL definiert. Die Unterschiede zu den in Abschnitt 2.1.3 entwickelten Datalog-Änderungs- und Transaktionskonzepten werden detailliert erörtert. Die Konsequenzen dieser Unterschiede für inkrementelle Änderungspropagierungsverfahren werden ausführlich in Kapitel 5 diskutiert. Die im folgenden verwendete SQL-Änderungssprache unterscheidet sind von der des SQL3-Standards nur geringfügig.

- Wie auch bereits bei den Anfragesprachen sind Duplikate ausgeschlossen.
- Einige Optionen der Anweisungen[41] des SQL3-Standards werden nicht berücksichtigt, da sie lediglich Kurzschreibweisen sind.
- In den Änderungsanweisungen sind SQL-Formeln (Definition 2.2.2) zugelassen, so daß die Einschränkungen aus der Basisfunktionalität der SQL-Anfragesprache resultieren.
- Die Änderungsanweisungen werden nur für Basisrelationen ausgeführt. Im SQL3-Standard sind Änderungsanweisungen für Sichten eingeschränkt zulässig. Änderbare und nicht änderbare Sichten[42] werden unterschieden. So sind z.B. nur die Attribute änderbar, die zu der Basisrelation gehören, deren Primärschlüssel auch die eindeutig identifizierende Attributkombination der Sicht darstellt. Nicht das Transformieren von Sichtänderungen auf die Basisfaktenänderungen ('view update') ist Gegenstand dieser Arbeit sondern der umgekehrte Schritt das Herleiten der aus Basisfaktenänderungen resultierenden Sichtänderungen. Zudem hat diese Restriktion keine Konsequenzen für das in Kapitel 5 entwickelte Verfahren.

Eine SQL-Änderungsanweisung ist eine der drei Anweisungen Einfügung (INSERT), Löschung (DELETE) und Modifikation (UPDATE). Für die nachfolgenden Definitionen sei eine SQL-Datenbank $\mathcal{D}^{\mathcal{S}} = \langle \mathcal{F}, \mathcal{S}^{\mathcal{S}} \rangle$ mit dem DB-Schema $\mathcal{S}^{\mathcal{S}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C}, \mathcal{T} \rangle$ und der Signatur $\Sigma^S = \langle\ \mathcal{PS}, \mathcal{AS}, \mathcal{CS}, \mathcal{TS}\ \rangle$ gegeben.

Definition 2.2.12 (SQL-Einfügeanweisung)
Sei $B \in rel(\mathcal{B})$ das Relationssymbol einer Basisrelation in einer SQL-DB $\mathcal{D}^{\mathcal{S}}$. Eine SQL-Einfügeanweisung ist ein Ausdruck der Form

INSERT INTO B ins_source,

wobei B die Basisrelation bezeichnet, in die Tupel eingefügt werden.
ins_source kann

1. *ein Ausdruck der Form VALUES $(\ t_1, \ldots, t_n\ )$*
   sein, wobei die Terme $t_1, \ldots, t_n$ Konstanten sind, die die Attributwerte des in B einzufügenden Tupels spezifizieren (primitive Einfügung). Die Attributstruktur von $t_1, \ldots, t_n$ muß vereinigungskonform zu der von B sein.
2. *oder ein SQL-Anfrageausdruck Q, dessen Ergebnismenge die Menge der einzufügenden Tupel darstellt (komplexe Einfügung), und die über eine zu B vereinigungskonforme Attributstruktur verfügt.*

[41] [ANSI99II] 14.6-14.10, [DD97] 9.3-9.5
[42] [ANSI99II] 4.16, 7, 11.21, [DD97], 13.3, [GP99] 18

Definition 2.2.13 (SQL-Modifikationsanweisung)
Sei $B \in rel(\mathcal{B})$ das Relationssymbol einer Basisrelation in einer SQL-DB $\mathcal{D}^{\mathcal{S}}$ mit den Attributen $A_1, \ldots, A_n$. Eine SQL-Modifikationsanweisung ist ein Ausdruck der Form

$$\begin{array}{ll} \textit{UPDATE} & B \ \textit{AS} \ T_B \\ \textit{SET} & A_k = \textit{upd\_source}_k, \ \ldots, \ A_l = \textit{upd\_source}_l \\ \textit{WHERE} & F, \end{array}$$

wobei $A_k, \ldots, A_l$ $(1 \leq k, l \leq n)$ Attribute von B sind, denen Werte von $\textit{upd\_source}_k, \ldots, \textit{upd\_source}_l$ zugewiesen werden. upd_source kann eine Konstante sein oder ein SQL-Anfrageausdruck Q, dessen Ergebnismenge aus genau einem Attribut besteht und genau ein Tupel zurückliefert. F ist eine SQL-Formel, mittels der die Menge der zu modifizierenden Tupel qualifiziert wird. Attributterme mit der Tupelvariable T_B treten in F auf.

Definition 2.2.14 (SQL-Löschanweisung)
Sei $B \in rel(\mathcal{B})$ das Relationssymbol einer Basisrelation in einer SQL-DB $\mathcal{D}^{\mathcal{S}}$. Eine SQL-Löschanweisung ist ein Ausdruck der Form

$$\textit{DELETE FROM} \ B \ \textit{AS} \ T_B \ \textit{WHERE} \ F,$$

wobei F eine SQL-Formel ist, mittels der die aus B zu löschenden Tupel qualifiziert werden. Attributterme mit der Tupelvariable T_B treten in F auf.

Ohne die Semantik zu modifizieren, aber um auch in diesem Kontext über eine TRC-konforme Formel-Syntax zu verfügen, sind die UPDATE- und DELETE-Anweisungen in SQL um die AS-Klausel zur Spezifikation der Tupelvariablen T_B ergänzt. Auf der Basis von T_B wird in F die Bedingung formuliert, die die in B zu modifizierenden/löschenden Tupel erfüllen müssen. Für die INSERT-Anweisung wird keine Tupelvariable eingeführt, da dort die Anfrage Q völlig unabhängig von der zu manipulierenden Basisrelation B definiert wird. Die UPDATE- und DELETE-Anweisungen stellen eine vereinfachte Schreibweise dar, bei der Ziel der Aktion und Quelle der zu ändernden Tupel zusammenfallen (B). Die WITH-Klausel gehört anders als bei Datalog zu der Anfrage Q bzw. zu den Anfrageausdrücken, die ggf. in der Formel F als Operanden auftreten. Um UPDATE- oder DELETE-Anweisungen mit der Semantik primitiver Änderungsanweisungen auszuführen, müssen die Bedingungen so formuliert werden, daß sie nur ein zu modifizierendes bzw. zu löschendes Tupel qualifizieren. Für eine prozedurale Steuerung der Ausführung mehrerer Änderungsanweisungen müssen diese in Prozeduren und Funktionen einer prozeduralen Programmiersprache eingebettet werden. Die syntaktischen Analogien und Unterschiede zwischen der Datalog- und der SQL-DML werden anhand von vier Beispielen deutlich.

Beispiel 2.2.15 (Vergleich Datalog und SQL)
Seien die Relationen p (3 Attribute) und q (2 Attribute) gegeben.

- *Datalog:* $+p(1, a, 4711)$

  SQL: $\textit{INSERT INTO} \ p \ \textit{VALUES} \ (1, a, 4711)$

- *Datalog:* $+p(X, 4711, X) \Leftarrow q(X, X)$

  SQL: *INSERT INTO* p *SELECT* $t_q.a_1, 4711, t_q.a_1$ *FROM* q *AS* t_q
  WHERE $t_q.a_1 = t_q.a_2$

- *Datalog:* $\pm(\ p(X, Y, 13), p(X, 4711, Z)\ ) \Leftarrow p(X, 4711, Z) \wedge q(Y, 5)$

  SQL: *UPDATE* p *AS* t_p
  SET $t_p.a_2 = (\textit{SELECT}\ t_q.a_1\ \textit{FROM}\ q\ \textit{AS}\ t_q\ \ \textit{WHERE}\ t_q.a_2 = 5)$,
  $t_p.a_3 = 13$
  WHERE $t_p.a_2 = 4711$

- *Datalog:* $-p(X, 5, Z) \Leftarrow p(X, 5, Z) \wedge Z > X$

  SQL: *DELETE FROM* p *AS* t_p *WHERE* $t_p.a_2 = 5$ *AND* $t_p.a_3 > t_p.a_1$

Die syntaktischen Einschränkungen der Änderungsanweisungen in SQL3 gewährleisten ebenso wie bei den Anfragen, daß syntaktisch korrekte SQL-Änderungsanweisungen auch bereichsbeschränkt sind. Wenn im weiteren der Begriff Änderungsanweisung verwendet wird, ist immer eine bereichsbeschränkte Änderungsanweisung gemeint.

In SQL3 wird eine einzelne DML-Anweisung genau wie in Datalog deterministisch ausgeführt. Diese Reihenfolgeunabhängigkeit gilt in Datalog auch für die Ausführung einer Menge von Änderungsanweisungen am Ende einer Transaktion, aber nicht in SQL. Eine SQL-Transaktion ist eine Folge von DML-Anweisungen[43] (Abbildung 2.4). Die Ordnung ergibt sich aufgrund der zeitlichen Abfolge, mit der ein Anwender die Befehle vom DB-System ausführen läßt. Jede Anweisung wird unmittelbar ausgeführt und nicht erst zum Transaktionsende mit der Konsequenz, daß nach jeder Änderungsanweisung ein neuer Zwischenzustand vorliegt. Eine Transaktion beginnt mit dem ersten DML-Befehl seit der letzten Transaktion und endet mit dem COMMIT-Befehl[44]. Obwohl die Reihenfolge der Anweisungen entscheidend bei der Ausführung einer SQL-Transaktion ist, wird wie in Datalog die Kurzschreibweise für eine Transaktion $\mathcal{M}$ eingeführt.

Definition 2.2.16 (SQL-Transaktion)
Seien $M_1, \ldots, M_n$ SQL-Änderungsanweisungen. Dann ist jede Folge von Änderungsanweisungen $M_1; \ldots; M_n;$, die mit dem Befehl COMMIT abschließt, eine Transaktion:

$M_1;\ M_2;\ \ldots\ M_n;$ *COMMIT;.* *kurz:* $\mathcal{M}$.

Aus diesem zu Datalog sehr verschiedenen Transaktionskonzept resultieren drei sehr grundlegende Eigenschaften von SQL3, die die Entwicklung SQL-basierter inkrementeller Verfahren in Kapitel 5 sehr wesentlich beeinflussen.

1. In SQL gibt es zwei Integritätsprüfungszeitpunkte. Die IMMEDIATE zu prüfenden Integritätsbedingungen werden unmittelbar nach der Ausführung einer Anweisung

[43] [ANSI99II] 4.32, 16, [DD97] 5.4, [GP99] 36

[44] Der ROLLBACK-Befehl wird in SQL nicht weiter berücksichtigt, da ein aufgrund eines Integritätsfehlers fehlgeschlagenes COMMIT-Ereignis die gleichen Konsequenzen hat.

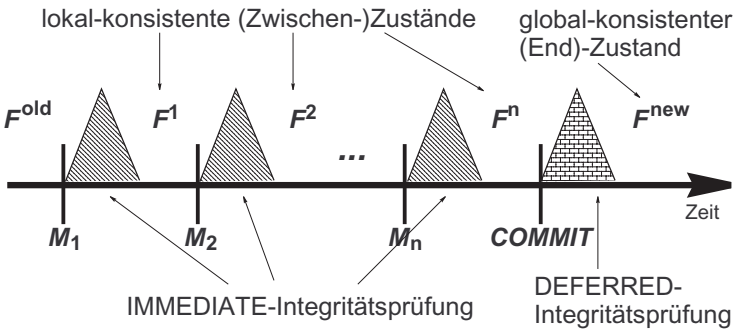

Abbildung 2.4: SQL-Transaktion

geprüft, DEFERRED-Integritätsbedingungen hingegen zum Ende der Transaktion (COMMIT-Ereignis). Bei einer reihenfolgeunabhängigen Ausführung von DML-Anweisungen wie in Datalog würde eine Prüfung unmittelbar nach der Ausführung einer Anweisung aufgrund des Nichtdeterminismus ein für Anwender nicht nachvollziehbares Ergebnis liefern.

2. Die Ausführung einer Änderungsanweisung stellt quasi eine 'Untertransaktion' (elementare Transaktion) dar, die einen lokal-konsistenten Zustand in einen wiederum lokal-konsistenten Zustand überführt. Die Ausführung einer einzelnen Änderungsanweisung erfolgt ebenso atomar wie die einer Transaktion (lokales und globales ROLLBACK).

3. Eine Nettoeffektberechnung bzgl. mehrerer Änderungsanweisungen ist nicht erforderlich, da Konfliktfälle nur bei 'quasi simultaner' Ausführung mehrerer Anweisungen auftreten können.

Vor diesem Hintergrund wird die Semantik von Änderungsanweisungen und -Transaktionen definiert. In SQL werden nur Änderungen ausgeführt, die die zugrunde liegende Faktenmenge effektiv manipulieren. Fakten, die bereits vorliegen, werden nicht nochmals eingefügt, ebenso wenig wie Fakten wiederholt gelöscht werden.

Definition 2.2.17 (Effektive SQL-Änderungsanweisung)
Sei $B \in rel(\mathcal{B})$ das Symbol einer SQL-Basisrelation aus $\mathcal{D}^{\mathcal{S}}$ mit den Attributen $A_1, \ldots, A_n$.

1. *Ist M eine primitive SQL-Einfügeanweisung*

   $$\textit{INSERT INTO } B \textit{ VALUES } (c_1, \ldots, c_n),$$

   wobei $c_1, \ldots, c_n$ Konstanten sind, dann ist die zugehörige effektive SQL-Änderungsanweisung M^{eff} ein Ausdruck der Form

   $$\textit{INSERT INTO } B \textit{ SELECT } c_1, ..., c_n \textit{ FROM } B \textit{ AS } T_B \textit{ WHERE } F^{eff}$$

   mit $F^{eff} :=$ $\textit{NOT EXISTS} (\textit{SELECT } A_1, ..., A_n \textit{ FROM } B \textit{ AS } T_{B'}$
   $\textit{WHERE } T_{B'}.A_1 = c_1 \textit{ ... AND } T_{B'}.A_n = c_n)$.

2. *Ist M eine komplexe SQL-Einfügeanweisung*

$$\begin{array}{l}\textit{INSERT INTO } B \textit{ SELECT } T_{R_i}.A_g, ..., T_{R_j}.A_h \\ \qquad\qquad\qquad\quad \textit{FROM } R_i \textit{ AS } T_{R_i}, ..., R_j \textit{ AS } T_{R_j} \textit{ WHERE } F,\end{array}$$

da ist die zugehörige effektive SQL-Änderungsanweisung M^{eff} ein Ausdruck der Form

$$\begin{array}{l}\textit{INSERT INTO } B \textit{ SELECT } T_{R_i}.A_g, \ldots, T_{R_j}.A_h \\ \qquad\qquad\qquad\quad \textit{FROM } R_i \textit{ AS } T_{R_i}, \ \ldots, \ R_j \textit{ AS } T_{R_j} \\ \qquad\qquad\qquad\quad \textit{WHERE } F \textit{ AND } F^{eff}\end{array}$$

mit $$\begin{array}{l}F^{eff} := \textit{NOT EXISTS } (\textit{SELECT } A_1, \ldots, A_n \textit{ FROM } B \textit{ AS } T_{B'} \\ \qquad\qquad\qquad\qquad\quad \textit{WHERE } T_{B'}.A_1 = T_{R_i}.A_g \ \ldots \\ \qquad\qquad\qquad\qquad\quad \textit{AND } \quad T_{B'}.A_n = T_{R_j}.A_h).\end{array}$$

3. *Ist M eine SQL-Löschanweisung, dann gilt für die zugehörige effektive SQL-Änderungsanweisung $M^{eff} := M$.*

4. *Ist M eine SQL-Modifikationsanweisung*

$$\begin{array}{l}\textit{UPDATE } B \textit{ AS } T_B \textit{ SET } A_g = upd\_source_g, ..., A_h = upd\_source_h \\ \textit{WHERE } F\end{array}$$

wobei $A_g, ..., A_h$ die durch M modifizierten Attribute von R sind und $A_i, ..., A_j$ die nichtmodifizierten Attribute ($\{A_g, ..., A_h\} \cap \{A_i, ..., A_j\} = \emptyset$). Dann ist die zugehörige effektive SQL-Änderungsanweisung M^{eff} ein Ausdruck der Form

$$\begin{array}{l}\textit{UPDATE } B \textit{ AS } T_B \textit{ SET } A_g = upd\_source_g, ..., A_h = upd\_source_h \\ \textit{WHERE } F \textit{ AND } F^{eff}\end{array}$$

mit $$\begin{array}{l}F^{eff} := \textit{NOT EXISTS } (\textit{SELECT } A_1, \ldots, A_n \textit{ FROM } B \textit{ AS } T_{B'} \\ \qquad\qquad\qquad\qquad\quad \textit{WHERE } T_{B'}.A_g = upd\_source_g ... \\ \qquad\qquad\qquad\qquad\quad \textit{AND } \quad T_{B'}.A_h = upd\_source_h \\ \qquad\qquad\qquad\qquad\quad \textit{AND } \quad T_{B'}.A_i = T_B.A_i \ ... \\ \qquad\qquad\qquad\qquad\quad \textit{AND } \quad T_{B'}.A_j = T_B.A_j \ ).\end{array}$$

Definition 2.2.18 (Menge effektiv betroffener Tupel)
Sei $\mathcal{D}^{\mathcal{S}} = \langle \mathcal{F}, \mathcal{S}^{\mathcal{S}} \rangle$ eine SQL-Datenbank und M eine SQL-Änderungsanweisung bzgl. der Basisrelation mit dem Symbol $B \in rel(\mathcal{B})$ und den Attributen $A_1, \ldots, A_n$. Die Bedingung einer effektiven Änderungsanweisung M^{eff} sei $F^* := F$ AND F^{eff}. $\{\sigma_i\}$ $(i \geq 0)$ sei die Menge der Grundsubstitutionen, die sich bei der Auswertung von F^* über $\mathcal{F}$ ergibt.
Dann ist $\mathcal{T}(M, \mathcal{F}) := \langle \mathcal{T}^-(M, \mathcal{F}), \mathcal{T}^+(M, \mathcal{F}), \mathcal{T}^{\pm}(M, \mathcal{F}) \rangle$ die Menge der durch die Ausführung von M bzgl. $\mathcal{F}$ betroffenen Tupel.

1. *Ist M eine INSERT-Anweisung, dann gilt*

$$\begin{array}{l}\mathcal{T}^+(M, \mathcal{F}) := \{ \ B(\ (A_1, \ldots, A_n)\sigma_i \ ) \mid eval(F^*\sigma_i, \mathcal{F}) = TRUE \ \} \\ \mathcal{T}^-(M, \mathcal{F}) := \emptyset \\ \mathcal{T}^{\pm}(M, \mathcal{F}) := \emptyset.\end{array}$$

2. *Ist M eine* DELETE*-Anweisung, dann gilt*

$$\begin{aligned}\mathcal{T}^{+}(M,\mathcal{F}) &:= \emptyset \\ \mathcal{T}^{-}(M,\mathcal{F}) &:= \{\ B(\ (A_1,\ldots,A_n)\sigma_i\ ) \mid \quad eval(F^*\sigma_i,\mathcal{F}) = \mathit{TRUE}\ \} \\ \mathcal{T}^{\pm}(M,\mathcal{F}) &:= \emptyset.\end{aligned}$$

3. *Ist M eine* UPDATE*-Anweisung, dann gilt*

$$\begin{aligned}\mathcal{T}^{+}(M,\mathcal{F}) &:= \emptyset \\ \mathcal{T}^{-}(M,\mathcal{F}) &:= \emptyset \\ \mathcal{T}^{\pm}(M,\mathcal{F}) &:= \{\ B(\ (upd\_source_1,\ldots,upd\_source_n)\sigma_i\ )\ , \\ &\qquad B(\ (A_1,\ldots,A_n)\sigma_i\ )\ ) \mid \ eval(F^*\sigma_i,\mathcal{F}^{old}) = \mathit{TRUE}\ \},\end{aligned}$$

wobei upd_source_i durch A_i zu ersetzen ist, für diejenigen Attribute A_i, denen in der SET*-Liste keine neuen Werte zugewiesen werden.*

4. $$\begin{aligned}\mathcal{T}^{\pm+}(M,\mathcal{F}) &:= \{\ B^{new} \mid (B^{new},B^{old}) \in \mathcal{T}^{\pm}(M,\mathcal{F})\ \}. \\ \mathcal{T}^{\pm-}(M,\mathcal{F}) &:= \{\ B^{old} \mid (B^{new},B^{old}) \in \mathcal{T}^{\pm}(M,\mathcal{F})\ \}.\end{aligned}$$

In SQL werden (anders als in der Datalog-DML) für eine Änderungsanweisung alle betroffenen Tupel ermittelt und dann unmittelbar manipuliert. Daher können keine Mengen zu manipulierender Basisfakten entstehen, die durch verschiedene Änderungsanweisungen initiiert worden sind. Für Basisfaktenänderungen treten somit keine Konflikte auf, wie sie im Rahmen der Datalog-DML ausführlich diskutiert wurden (Abschnitt 2.1.3). Daher ist in SQL die zuvor definierte Menge effektiv betroffener Tupel $\mathcal{T}(M,\mathcal{F})$ einer Änderungsanweisung M zugleich der Nettoeffekt $\Delta(M,\mathcal{F})$, der durch die Ausführung von M bzgl. der Basisfaktenmenge $\mathcal{F}$ verursacht wird. Ist der Kontext der Änderungsanweisung und der Faktenmenge $(\mathcal{M},\mathcal{F})$ eindeutig, so wird auch die Kurzschreibweise $\Delta := \langle\ \Delta^+,\Delta^-,\Delta^{\pm}\ \rangle$ verwendet.

Definition 2.2.19 (Nettoeffekt einer SQL-Änderungsanweisung)
Sei $\mathcal{F}$ die Basisfaktenmenge von $\mathcal{DS}$ und M eine SQL-Änderungsanweisung mit der Menge der durch M effektiv betroffenen Tupel $\mathcal{T}(M,\mathcal{F})$, dann ist der Nettoeffekt der Änderungsanweisung M definiert als

$$\Delta(M,\mathcal{F}) := \mathcal{T}(M,\mathcal{F}) \qquad\qquad \Delta^*(M,\mathcal{F}) := \mathcal{T}^*(M,\mathcal{F})$$

wobei $* \in \{^{+},^{-},^{\pm},^{\pm+},^{\pm-}\}$.

Der Nettoeffekt einer SQL-Änderungsanweisung $\Delta(M,\mathcal{F})$ beschreibt den durch die Ausführung einer Änderungsanweisung M verursachten Zustandsübergang der alten Basisfaktenmenge $\mathcal{F}^i$ in die neue Faktenmenge $\mathcal{F}^{i+1}$. Alle durch M betroffenen Fakten (Δ-Mengen) werden deterministisch manipuliert. An die Ausführung einer Änderungsanweisung schließt sich unmittelbar die Prüfung der IMMEDIATE-Integritätsbedingungen an, die von dieser Anweisung betroffen sind (Abbildung 2.4). Wird für eines der zu manipulierenden Tupel ein Integritätsfehler erkannt, so werden nur alle Änderungen der aktuellen Anweisung zurückgerollt. Gehen aus dem Zusammenhang sowohl die Änderungsanweisung M wie auch die Faktenmenge $\mathcal{F}^i$ eindeutig hervor, so wird die Kurzbezeichnung $\mathcal{F}^{i+1}$ verwendet.

Definition 2.2.20 (Lokal-konsistenter Zwischenzustand)
Sei $\mathcal{F}^i$ eine konsistente Basisfaktenmenge und $\mathcal{C}^{IMM} \subseteq \mathcal{C}$ die Menge der IMMEDIATE zu prüfenden Integritätsbedingungen von $\mathcal{DS}$. M sei eine Änderungsanweisung, die angewendet auf $\mathcal{F}^i$ den Nettoeffekt

$$\Delta(M, \mathcal{F}^i) := \langle \Delta^+(M, \mathcal{F}^i), \Delta^-(M, \mathcal{F}^i), \Delta^{\pm}(M, \mathcal{F}^i) \rangle$$

ergibt. Der Effekt der Ausführung von M auf $\mathcal{F}^i$ ist der neue lokal-konsistente (Zwischen-) Zustand $\mathcal{F}^{i+1}$:

$$\mathcal{F}^{i+1}(\mathcal{F}^i, M) := \begin{cases} \mathcal{F}^i \cup (\Delta^+(M, \mathcal{F}^i) \cup \Delta^{\pm+}(M, \mathcal{F}^i)) \setminus (\Delta^-(M, \mathcal{F}^i) \cup \Delta^{\pm-}(M, \mathcal{F}^i)) \\ \qquad\qquad \textit{falls } \forall\, C \in \mathcal{C}^{IMM} \mid eval(C, \mathcal{F}^{i+1}) = \textit{TRUE} \\ \mathcal{F}^i \qquad\qquad \textit{sonst.} \end{cases}$$

Der Begriff des lokal-konsistenten Zwischenzustands basiert darauf, daß die DEFERRED-Integritätsbedingungen zu diesem Zeitpunkt noch nicht geprüft sind. Erst wenn auch die DEFERRED-Bedingungen am Ende der Transaktion evaluiert worden sind, liegt wieder ein (global-)konsistenter Zustand vor, der alle Anforderungen erfüllt. Die Ausführung einer Transaktion erzeugt, ausgehend von einem (global-)konsistenten Zustand $\mathcal{F}^{old}$, eine Folge lokal-konsistenter Zwischenzustände (Abbildung 2.4).

Definition 2.2.21 ((Global-)konsistenter Zustand)
Sei $\mathcal{F}^{old}$ eine konsistente Basisfaktenmenge und $\mathcal{C}^{DEF} \subseteq \mathcal{C}$ die Menge der DEFERRED zu prüfenden Integritätsbedingungen von $\mathcal{DS}$. $\mathcal{M}$:= M_1;...;M_n;COMMIT; sei eine Transaktion, die angewendet auf $\mathcal{F}^{old}$ die Folge $\mathcal{F}^1; \mathcal{F}^2; ...; \mathcal{F}^{new}$; lokal-konsistenter Zwischenzustände erzeugt.
Der Effekt der Ausführung von $\mathcal{M}$ auf $\mathcal{F}^{old}$ ist der neue konsistente (End-)Zustand $\mathcal{F}^{new}$.

$$\mathcal{F}^{new} := \begin{cases} \mathcal{F}^{new}(\mathcal{F}^{n-1}(\ ...\ \mathcal{F}^1(\mathcal{F}^{old}, M_1)\ ...\ ), M_n) \\ \qquad\qquad \textit{falls } \forall\, C \in \mathcal{C}^{DEF} \mid eval(C, \mathcal{F}^{new}) = \textit{TRUE} \\ \mathcal{F}^{old} \qquad\qquad \textit{sonst.} \end{cases}$$

Sind aus dem Kontext sowohl die Transaktion $\mathcal{M}$ wie auch die Ausgangsfaktenmenge $\mathcal{F}^{old}$ offensichtlich, so wird für den konsistenten (End-)Zustand $\mathcal{F}^{new}(\mathcal{F}^{old}, \mathcal{M})$ nur die Kurzbezeichnung $\mathcal{F}^{new}$ verwendet.

2.2.5 Trigger

In diesem Abschnitt werden zwei SQL-basierte Triggerkonzepte vorgestellt: Starburst- und SQL3-Trigger. Mittels Starburst-Triggern haben Ceri/Widom ([CW90], [CW91], [CW94]) bereits Anfang der 90er Jahre inkrementelle Verfahren zur Sichtenaktualisierung und Integritätsprüfung entwickelt und die Implementierungsansätze in Kapitel 5 basieren auf SQL3-Triggern.

Starburst-Trigger:

Starburst war ein Forschungsprojekt am IBM Almaden Research Center, in dessen Rahmen prototypische Funktionen für SQL-basierte DB-Systeme entwickelt wurden. Eine

von IBM entwickelte DBS-Komponente ist die Triggerkomponente 'Starburst Rule System' ([CW90], [CW91], [Wid91], [CW94] [Grie97]). In den verschiedenen Publikationen sind aufgabenspezifisch unterschiedliche Varianten des Triggerkonzepts definiert. Die Starburst-Trigger werden nur insoweit vorgestellt, wie sie von Ceri/Widom zur Implementierung inkrementeller Verfahren in [CW90], [CW91], [CW94] verwendet werden.

Definition 2.2.22 (Starburst-Trigger)
Ein Starburst-Trigger ist eine ECA-Regel der Form

$$G \quad \textit{WHEN} \quad E_1, \ldots, E_n$$
$$[\ \textit{IF}\ \ F\ ] \quad \textit{THEN}\ \ M_1;\ \ldots;\ M_g; \quad \textit{PRECEDES}\ \ G_1, \ldots, G_h,$$

wobei $G, G_1, ..., G_h$ Triggersymbole sind. E_i $(1 \leq i \leq n)$ ist eines der Ereignissymbole INSERTED INTO B, DELETED FROM B, UPDATED $B.A_j$. *A_j ist das Attributsymbol eines Attributs der Basisrelation B. F ist eine SQL-Formel und M_k $(1 \leq k \leq g)$ ist eine SQL-Änderungsanweisung.*

Mit einem CREATE RULE-Befehl wird ein Trigger als persistentes Schemaobjekt erzeugt. Ein Starburst-Trigger kann für verschiedene Ereignisse feuern und somit für verschiedene Basisrelationen definiert werden[45]. Um einen Trigger zu aktivieren, ist es ausreichend, wenn eines der Ereignisse aus der Liste $E_1, \ldots, E_n$ eintritt. Als Ereignis wird die Ausführung der Änderungsanweisungen für Einfügungen, Löschungen und Modifikationen erkannt. Ein Starburst-Trigger wird nicht unmittelbar bei Eintritt des Ereignisses gefeuert sondern erst zum 'rule assertion point', der immer zum Ende einer Transaktion oder auch vom Anwender während der Transaktion gesetzt wird.

Zum 'rule assertion point' wird von der Triggerkomponente der Nettoeffekt der Ereignisse deterministisch ermittelt (Nettoeffekt der Basisfaktenänderungen), die seit dem letzten 'rule assertion point' eingetreten sind. Regeln deterministischer Nettoeffektberechnungen sind ausführlich in Abschnitt 2.1.3 erörtert worden. Für z.B. die Einfügung und Löschung des gleichen Fakts bedeutet dies, daß zum 'rule assertion point' keine Ereignisse vorliegen, für die Trigger aktiviert werden können. Für die Nettoeffektänderungen werden von der Triggerkomponente die Trigger mengenorientiert gefeuert und ausgeführt. Mengenorientiert heißt in diesem Zusammenhang, daß ein Trigger für die Ausführung einer Anweisung einmal feuert, unabhängig davon, wie viele Fakten geändert wurden.

Beim Aktivieren eines Triggers werden keine Variablenbindungen für die optionale Bedingung (IF) und die Aktionen (THEN) des Triggers erzeugt (keine Instanziierung). Innerhalb der Aktionen und der Bedingung kann auf den neuen DB-Zustand der Relationen zugegriffen werden. Die Menge der manipulierten Fakten einer Relation liegen in systemintern verwalteten Δ-Mengen ('transition tables': INSERTED/DELETED / OLD UPDATED/NEW UPDATED B) vor. Auf die Transitionsrelationen kann im Aktions- und Bedingungsteil des Triggers lesend zugegriffen werden.

Die optionale PRECEDES-Klausel bietet die Möglichkeit, eine partielle Ordnung für eine Menge von Triggern zu definieren (Prioritäten). Der Trigger G wird vor den Triggern $G_1, \ldots, G_h$ ausgeführt. Sind keine Prioritäten definiert, feuern die Trigger in beliebiger

[45] In [Wid91] wird eine Starburst-Variante definiert, die sich wie bei SQL nur auf eine Relation bezieht.

Reihenfolge. Sind alle relevanten Trigger für ein Ereignis E_i gefeuert, wählt die Triggerkomponente den Trigger G aus, der über keinen aktivierten Vorgänger verfügt. Bevor G ausgeführt wird, wird die Bedingung F der IF-Klausel ausgewertet. Ist F nicht erfüllt, so wird G nicht ausgeführt, und der nächste aktivierte Trigger wird ausgewählt. Bricht die Ausführung einer triggerausgeführten Aktion fehlerhaft ab, so wird die gesamte Transaktion zurückgerollt.

SQL3-Trigger:

Obwohl aktive Regeln bereits in vielen kommerziellen DB-Systemen (DB/2, Oracle, Informix, MS SQL-Server etc.) seit Jahren implementiert sind, ist erst im Rahmen von SQL3 ein standardisiertes Konzept definiert worden ([ANSI99II] 4.35, 11.38, [CPM96], [GP99] 24, [WC96] 9[46], [Pat99] 10[47]). Die Unterschiede der im folgenden definierten Trigger gegenüber denen des SQL3-Standards sind nur geringfügig und beeinflussen nicht das Triggerausführungsmodell. Lediglich die Formel des Bedingungsteils darf nur eine SQL-Formel sein, und im Aktionsteil sind nur SQL-Anfragen und -Änderungsanweisungen zugelassen. Aufgrund der Geringfügigkeit der Unterschiede werden die Trigger im weiteren als SQL-Trigger bezeichnet.

Definition 2.2.23 (SQL-Trigger)
Ein SQL-Trigger ist eine ECA-Regel der Form

$$
\begin{array}{ll}
G & \textit{BEFORE} \mid \textit{AFTER} \quad E \quad \textit{ON} \quad B \\
& [\ \textit{REFERENCING} \quad \textit{OLD} \mid \textit{NEW} \quad \textit{ROW AS} \quad T^{row} \\
& \qquad\qquad\qquad\quad \textit{OLD} \mid \textit{NEW} \quad \textit{RELATION AS} \quad T^{rel}\ ] \\
& \textit{FOR EACH} \quad \textit{ROW} \mid \textit{STATEMENT} \\
& [\ \textit{WHEN} \quad (\ F\ )\ ] \quad \textit{BEGIN ATOMIC} \quad M_1;\ \ldots;\ M_n;\ \textit{END},
\end{array}
$$

wobei G das Triggersymbol ist und E eines der Ereignissymbole INSERT, DELETE *oder* UPDATE *[OF $A_1, \ldots, A_m$]. B ist eine Basisrelation und $A_1, \ldots, A_m$ bezeichnen Attribute von B. T^{row} ist eine Tupelvariable und T^{rel} ein Relationssymbol. F ist eine SQL-Formel und M_i ist eine SQL-Anfrage oder -Änderungsanweisung* $(1 \leq i \leq n)$.

Mit einem CREATE TRIGGER-Befehl wird ein Trigger als persistentes Schemaobjekt definiert. Ein Trigger wird für genau eine Basisrelation erzeugt (ON B). Je Trigger wird genau ein feuerndes Ereignis spezifiziert, welches eine Einfügung, Löschung oder Modifikation ist. Für eine Modifikation können Attribute spezifiziert werden, von denen zumindest bei einem eine Wertänderung gegeben sein muß. BEFORE und AFTER bezeichnen die beiden Zeitpunkte, zu denen der Trigger relativ zum feuernden Ereignis ausgeführt werden kann ('trigger action time'). Die Granularität der Ausführung eines aktivierten Triggers (mengen-/instanzorientiert) wird mit der FOR EACH ROW | STATEMENT-Klausel spezifiziert. Mittels der optionalen REFERENCING-Klausel werden an die Transitionsvariablen bzw. an die Transitionsrelationen[48] Tupelvariablen T^{row} bzw. Relationssymbole T^{rel} gebunden. Die Formel F der optionalen WHEN-Klausel ist die Ausführungsbedingung, und

[46] Widom/Ceri gehen zum Zeitpunkt der Veröffentlichung (1996) von einem Triggerkonzept aus, das in dieser Form nicht verabschiedet wurde. Das SQL3-Triggerkonzept weist eine wesentlich eingeschränktere Funktionalität als das dort diskutierte auf.

[47] Autoren sind Kulkarni, K., Mattos, N., Cochrane, R.

[48] Diese Klausel heißt in SQL3: REFERENCING ... OLD | NEW TABLE ([ANSI99II] 11.38).

die Folge der SQL-Anfragen und -Änderungsanweisungen $M_1; \ldots; M_n$; sind die Aktionen des Triggers. Unabhängig von der Granularität der Trigger (ROW/STATEMENT) wird die Reihenfolge der Ausführung von Triggern zum gleichen Zeitpunkt von der Triggerkomponente anhand der Erzeugungszeitpunkte der Trigger bestimmt. Ist Trigger G_1 vor G_2 erzeugt worden, so wird G_1 vor G_2 ausgeführt.

Mit den Begriffen **ROW-** und **STATEMENT-Trigger** werden in SQL unterschiedlichen Ausführungsarten bezeichnet. STATEMENT-Trigger werden einmal für jede feuernde Änderungsanweisung ausgeführt unabhängig davon wie viele Fakten betroffen sind (mengenorientiert). Sie werden auch ausgeführt, wenn kein Fakt manipuliert wird. Ein ROW-Trigger wird für jedes betroffene Fakt einmal ausgeführt (instanzorientiert).

Abhängig von der Ausführungsart sind entweder Transitionsvariablen oder Transitionsrelationen bei der Triggerausführung verfügbar. Sie enthalten zum Ausführungszeitpunkt die alten/neuen Werte eines manipulierten Fakts bzw. eine Menge manipulierter Fakten. Ihre Attributstruktur ist identisch zu der der Basisrelation B. Um auf die Transitionsvariablen und -relationen im Aktions- oder Bedingungsteil zugreifen zu können, müssen ihnen in der REFERENCING-Klausel Tupelvariablen T^{row} bzw. Relationssymbole T^{rel} zugeordnet sein. Die Transitionsvariablen und -relationen sind nur eingeschränkt in den verschiedenen Triggertypen verfügbar. Die Transitionsvariablen sind ausschließlich in ROW-Triggern verfügbar. Bei INSERT-Triggern sind weder OLD ROW noch OLD RELATION zugreifbar und bei DELETE-Triggern weder NEW ROW noch NEW RELATION. Transitionsvariablen und -relationen stellen die Grundlage dar, um im Triggeraktionsteil die Formeln transitionaler Integritätsbedingungen formulieren zu können.

BEFORE- und **AFTER-Trigger** unterscheiden sich nicht nur hinsichtlich des Ausführungszeitpunkts vor oder nach der Ausführung der Änderungsanweisung, die den Trigger feuert. Als Aktionen sind in AFTER-Triggern auch Anfragen und Änderungsanweisungen zugelassen[49]. In BEFORE-Triggern sind hingegen keine Änderungsanweisungen erlaubt, wohl aber Zuweisungen zu den NEW-Transitionsvariablen, um noch die einzufügenden, zu modifizierenden Werte verändern zu können. Da zum Zeitpunkt der BEFORE-Triggerausführungen die aktivierende Änderungsanweisung noch nicht ausgeführt ist, kann zudem im Aktions- und Bedingungsteil eines BEFORE-Triggers nicht auf die Transitionsrelation OLD | NEW RELATION zugegriffen werden. Grund für diese unterschiedliche Funktionalität sind die unterschiedlichen Ausführungsmodelle (s.u., [CPM96], [Pat99] 10.2.3). Neben dem Ausführungszeitpunkt motivieren diese beiden Restriktionen die unterschiedliche Verwendung beider Triggerarten. BEFORE-Trigger werden in erster Linie zur Prüfung statischer und transitionaler Integritätsbedingungen verwendet und bieten die Möglichkeit, die zu manipulierenden Fakten zu korrigieren. AFTER-Trigger können neben der Integritätsprüfung insbesondere auch für Folgeverarbeitungen verwendet werden.

Die **Semantik der SQL3-Trigger**, wie sie in [ANSI99II] 4.35 definiert ist, wird unverändert übernommen. Cochrane et.al. erklären in [CPM96] das Triggerausführungsmodell sehr anschaulich. Insbesondere die Problematik der Ausführung von Triggern, die durch triggerausgeführte Änderungen aktiviert werden, und die Integration der Integritätsprüfung in den Prozeß der Triggerausführungen werden detailliert erörtert.

[49] In SQL3 sind zudem die Aufrufe benutzerdefinierter Funktionen/Prozeduren zugelassen.

Wird eine Änderungsanweisung INSERT, UPDATE, DELETE für eine Basisrelation ausgeführt, so ermittelt die Triggerkomponente die zugehörigen Trigger. Die gefeuerten BEFORE-Trigger werden gemäß ihrer Triggererzeugungszeit in die BEFORE-Triggerausführungsliste des Ereignisses (BEFORE-Liste) eingeordnet und die gefeuerten AFTER-Trigger in die AFTER-Liste des Ereignisses. Vor der Ausführung eines Triggers wird die Formel der WHEN-Klausel angewendet. Wann genau die WHEN-Bedingung geprüft wird, ob bei der Aktivierung oder vor der Ausführung, geht aus den Publikationen des ANSI-Standardisierungsgremiums nicht hervor. Cochrane et.al. definieren sie in [CPM96] als eine Ausführungsbedingung. Ist sie erfüllt, werden die Aktionen im Aktionsteil der Regel ausgeführt. Alle aktivierten BEFORE-Trigger werden vor dem Ereignis ausgeführt und alle gefeuerten AFTER-Trigger im Anschluß an die für alle relevanten Fakten ausgeführten Datenmanipulationen und IMMEDIATE-Integritätsprüfungen.

Hinsichtlich der Semantik von Triggern muß zwischen nichttriggerausgeführten und triggerausgeführten Änderungen unterschieden werden, da die durch sie gefeuerten AFTER-Trigger sehr unterschiedlich behandelt werden. Die Ausführung einer nichttriggerausgeführten Änderung impliziert (in dieser Reihenfolge) die Ausführung der BEFORE-Trigger, die Manipulation der betroffenen Fakten, die Integritätsprüfung und anschließend die Ausführung der AFTER-Trigger. Die von AFTER-Triggern ausgeführten Änderungen sind Aktionen, die von der ursprünglichen nichttriggerausgeführten Änderung abhängig sind. Die Triggerkomponente verwaltet für jedes Ereignis eigene BEFORE-Listen, die vollständig abgearbeitet sein müssen, bevor die Anweisung ausgeführt werden kann. AFTER-Listen aber werden ausschließlich für nichttriggerausgeführte Ereignisse erzeugt und verwaltet, so daß die durch triggerausgeführte Ereignisse gefeuerten AFTER-Trigger in die AFTER-Liste des ursprünglichen nichttriggerausgeführten Ereignisses eingeordnet werden. Die Ausführung einer triggerausgeführten Änderung beinhaltet somit (in dieser Reihenfolge) nur folgende Aktionen: die Ausführung der BEFORE-Trigger, die Manipulation der betroffenen Fakten, die Integritätsprüfung, die Einordnung der AFTER-Trigger in die AFTER-Liste des ursprünglichen Ereignisses. Anschließend wird mit der Anweisung fortgefahren, die der aktuellen im Triggeraktionsteil folgt.

Die zeitliche Abfolge der Aktionen bei der Ausführung einer nichttriggerausgeführten Änderungsanweisung M mit abhängigen Änderungen kann wie nachfolgend zusammengefaßt werden. Formal definiert wird das Ausführungsmodell zum Abschluß der Erörterungen in Definition 2.2.24. $\mathcal{BL}_M, AL_M$ seien die BEFORE- und AFTER-Listen von M. Seien $M_1, ..., M_n$ abhängige Änderungsanweisungen mit den BEFORE-Listen $\mathcal{BL}_{M_1}, ..., \mathcal{BL}_{M_n}$. Die aktivierten Trigger mit der kleinsten Erzeugungszeit stehen zu Beginn der Liste.

1. Die Triggerkomponente ermittelt für M die betroffenen Fakten und die zu feuernden BEFORE- und AFTER-Trigger und ordnet sie in $\mathcal{BL}_M, AL_M$ ein.

2. Alle BEFORE-Trigger aus $\mathcal{BL}_M$ werden ausgeführt. Aufgrund der zulässigen Aktionen können nur den Transitionsvariablen neue Werte zugewiesen werden.

3. Alle durch M betroffenen Fakten werden manipuliert und ihre Konsistenz geprüft.

4. Der erste Trigger G_1^{After} aus $\mathcal{AL}_M$ mit den Aktionen $M_1, ..., M_n$ wird ausgeführt. Die Änderungsanweisung M_1 wird ausgeführt.

(a) Die Triggerkomponente ermittelt für M_1 die betroffenen Fakten und die zu feuernden BEFORE- und AFTER-Trigger und ordnet sie in $\mathcal{BL}_{M_1}$ und $\mathcal{AL}_M$ ein.

(b) Alle BEFORE-Trigger aus $\mathcal{BL}_{M_1}$ werden ausgeführt.

(c) Alle durch M_1 betroffenen Fakten werden manipuliert und ihre Konsistenz geprüft.

(d) Die Schritte (a) bis (c) werden für die übrigen Änderungsanweisungen $M_2, ..., M_n$ von G_1^{After} ausgeführt.

5. Schritt 4 wird solange wiederholt, bis die Liste $\mathcal{AL}_M$ leer ist, dann erst ist die Ausführung von M beendet und der neue Zustand nach Ausführung der Basisfaktenänderung mit allen triggerausgeführten Änderungsanweisungen $\mathcal{F}^{new}$ liegt vor.

Motiviert ist dieses Ausführungsmodell, wie auch die Restriktionen bei den Aktionen in BEFORE-Triggern, durch die Notwendigkeit, die aus SQL2 bekannte, nichtdeterministische Integritätsprüfung bei FOREIGN KEY-Constraints mit der Fehlerkorrektur-Option (ON UPDATE | DELETE) auch weiterhin zu gewährleisten. Wenn SQL3-FOREIGN KEYs zugelassen sind, muß obiger Algorithmus um Triggerausführungen erweitert werden, die durch die Ausführung von Fehlerkorrektur-Anweisungen ausgelöst werden. Für alle im Rahmen der Integritätsprüfung (Aktionen 3. und 4.(c)) ausgeführten Fehlerkorrekturen werden dann die Schritte 4.(a) - (c) ausgeführt, um die Trigger zu ermitteln, die BEFORE-Trigger auszuführen, die Fakten zu manipulieren und die AFTER-Trigger in die AFTER-Liste des Ursprungsereignisses einzuordnen. Anschließend wird die Schleife 5. solange wiederholt, bis alle AFTER-Trigger (auch die durch FOREIGN KEY-Änderungsanweisungen aktivierten) ausgeführt sind. Trigger- und FOREIGN KEY-ausgeführte Änderungsanweisungen werden im Ausführungsmodell gleich behandelt, beides sind abhängige Ereignisse. Da mittels BEFORE-Triggern die Faktenmenge nicht manipuliert werden kann, wird somit die Reihenfolgeunabhängigkeit auch bei abhängigen Aktionen nicht gefährdet. Detailliert diskutiert wird diese Problematik in [CPM96], [Pat99] 10.2.

Löst eine Änderungsanweisung M eine Folge von Triggerausführungen mit weiteren Änderungsanweisungen aus, und schlägt die Integritätsprüfung einer dieser Änderungsanweisungen fehl, oder terminiert die Ausführung einer dieser Trigger fehlerhaft (Abbruch), so werden alle Manipulationen einschließlich der der auslösenden Anweisung M zurückgerollt. Die Ausführung von M hat in diesem Fall keinen Effekt bzgl. der Faktenmenge ($\mathcal{F}^{new} := \mathcal{F}^{old}$). Eine Änderungsanweisung bildet zusammen mit den durch sie gefeuerten Triggern und allen durch diese Trigger ausgeführten Anweisungen eine atomare Einheit.

Der $\oplus$-Operator sei ein Operator zur Einordnung eines Triggers gemäß seiner Erzeugungszeit in eine Trigger-Liste und der $\ominus$-Operator ein Operator für die Entfernung des ersten Objekts einer Liste von Triggern oder Aktionen. Die Ausführung eines Triggers G_i auf einer Faktenmenge $\mathcal{F}^{old}$ wird mit $G_i(\mathcal{F}^{old}, \Delta_M^*)$ bezeichnet, wobei $* \in \{^+, ^-, ^\pm\}$ die Änderungsart bezeichnet (Einfügung, Löschung, Modifikation) und Δ_M^* die Transitionsrelationen der durch eine Änderungsanweisung M manipulierten Basisrelation. Die Zeilennummern des Algorithmus entsprechen den Punkten obiger Auflistung. In der Abbildung 5.4 werden die Abläufe und Zusammenhänge des Ausführungsmodells nochmals veranschaulicht. Da ein Detail problematisch hinsichtlich der Implementierung von Integritätsprüfungsverfahren ist, ist die Abbildung in Kapitel 5 zu finden.

Definition 2.2.24 (SQL-Triggerausführungsmodell)
Sei M eine nichttriggerausgeführte Änderungsanweisung und $\mathcal{BL}_M, AL_M$ ihre BEFORE- *und* AFTER-*Listen. G^{Before}, G^{After} seien* BEFORE- *und* AFTER-*Trigger einer SQL-DB $\mathcal{D}^{\mathcal{S}}$. $\mathcal{A}_j := [M_{j1}, ..., M_{jm}]$ sei die Liste der Aktionen eines G_j^{After}-Triggers.*
$\mathcal{F}^{old}/\mathcal{F}^{new}$ seien die Zustände vor und nach der Ausführung von M. $\Delta^\star$ seien die Transitionsrelationen der manipulierten Relationen ($* \in \{^+, ^-, ^\pm\}$, $\star \in \{^+, ^-, ^\pm, ^{\pm+}, ^{\pm-}\}$).

BEGIN
-- Ausführen des Ursprungsereignisses M
-- Ermitteln der zu manipulierenden Fakten
(1.) *Berechnen der $\Delta_M^\star$-Fakten auf $\mathcal{F}^{old}$;*
-- Aktivieren der betroffenen Trigger und Einordnen in die Triggerlisten
(1.) $\mathcal{BL}_M := [\ G_i^{Before} \mid G_i^{Before}$ *wird durch M gefeuert* $]$;
(1.) $\mathcal{AL}_M := [\ G_j^{After} \mid G_j^{After}$ *wird durch M gefeuert* $]$;
-- Ausführen der BEFORE*-Trigger und der Änderungen von M*
(2.) $\Delta_M^\star := G_i^{Before}(\mathcal{F}^{old}, \Delta_M^\star)$; *(für alle $G_i^{Before} \in \mathcal{BL}_M$)*
(3.) $\mathcal{F}' := \mathcal{F}^{old} \cup (\Delta_M^+ \cup \Delta_M^{\pm+}) \setminus (\Delta_M^- \cup \Delta_M^{\pm-})$;

-- Ausführen aller direkt und indirekt abhängigen AFTER*-Trigger*
(5.) *REPEAT*
-- Ausführen des aktivierten Triggers mit der kleinsten Priorität G_j^{After}
(4.) $\mathcal{AL}_M := \mathcal{AL}_M \ominus G_j^{After}$;
-- Ausführen aller Aktionen eines Triggers G_j^{After}
(d) *REPEAT*
-- Ausführen der nächsten Triggeraktion M_{jk} von G_j^{After}
(a) $\mathcal{A}_j := \mathcal{A}_j \ominus M_{jk}$;
(a) *Berechnen der $\Delta_{M_{jk}}^\star$-Fakten auf $\mathcal{F}'$;*
-- Aktivieren der betroffenen Trigger und Einordnen in die Triggerlisten
(a) $\mathcal{BL}_{M_{jk}} := [\ G_{jkg}^{Before} \mid G_{jkg}^{Before}$ *wird durch M_{jk} gefeuert* $]$;
(a) $\mathcal{AL}_M := \mathcal{AL}_M \oplus [\ G_{jkh}^{After} \mid G_{jkh}^{After}$ *wird durch M_{jk} gefeuert* $]$;
-- Ausführen der Änderungen von M_{jk}
(b) $\Delta_{M_{jk}}^* := G_{jkg}^{Before}(\mathcal{F}^{old}, \Delta_{M_{jk}}^*)$; *(für alle $G_{jkg}^{Before} \in \mathcal{BL}_{M_{jk}}$)*
(c) $\mathcal{F}^{M_{jk}} := \mathcal{F}' \cup (\Delta_{M_{jk}}^+ \cup \Delta_{M_{jk}}^{\pm+}) \setminus (\Delta_{M_{jk}}^- \cup \Delta_{M_{jk}}^{\pm-})$;
(c) $\mathcal{F}' := \mathcal{F}^{M_{jk}}$
(d) *UNTIL* $\mathcal{A}_j = \emptyset$;
(5.) *UNTIL* $\mathcal{AL}_M = \emptyset$;
(5.) $\mathcal{F}^{new} := \mathcal{F}'$;
END.

Dieses Triggerkonzept ist bereits im DB/2 DB-System ([IBM00], [CPM96]) ohne syntaktische oder semantische Abweichungen implementiert worden. Die Oracle-Trigger ([Ora99] 7, [WC96] 9.3.1, [GW99] 20) weisen nur geringfügige syntaktische Abweichungen vom SQL/SQL3-Konzept auf. Die WHEN-Bedingung kann keine Anfragen enthalten, und ein Trigger kann für mehrere Ereignisse definiert werden. Semantisch werden die Unterschiede gerade hinsichtlich der zuvor diskutierten Prüfung von FOREIGN KEY-Constraints deutlich. Bzgl. der Aktionen in BEFORE-Triggern sind keine Restriktionen definiert. Die Trigger werden unmittelbar vor oder nach dem feuernden Ereignis ausgeführt. Die

Ausführung einer Änderungsanweisung beginnt mit einem BEFORE STATEMENT-Trigger. Für jedes betroffene Fakt werden unmittelbar vor der Manipulation die BEFORE ROW-Trigger ausgeführt und unmittelbar danach die AFTER ROW-Trigger. Die Ausführung der Anweisung schließt mit einem AFTER STATEMENT-Trigger ab. Konsequenz dieses Modells ist das sogenannte 'mutating table'-Problem. Damit die Reihenfolgeunabhängigkeit der Ausführung einzelner Änderungsanweisungen gewährleistet bleibt, besteht die Restriktion, daß innerhalb von ROW-Triggern nicht auf die aktuell zu ändernde Relation zugegriffen werden kann.

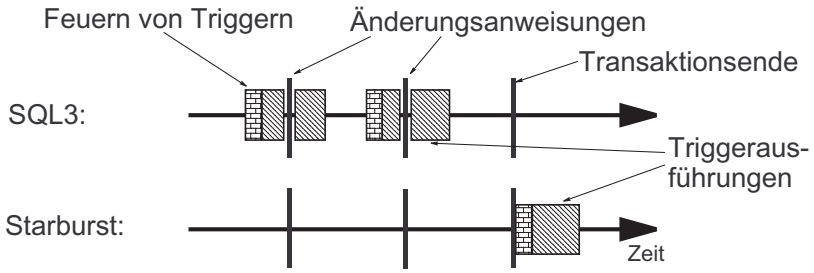

Abbildung 2.5: SQL- und Starburst-Triggerausführungsmodelle

Aufgrund ihrer Relevanz für die Implementierung inkrementeller Verfahren werden wesentliche Unterschiede der Ausführungsmodelle der Starburst- und SQL3-Trigger in der Abbildung 2.5 skizziert.

Kapitel 3

Normalisiertes SQL

Ausgehend von dem in Abschnitt 2.2 definierten Basis-SQL3 wird in diesem Abschnitt unter dem Gesichtspunkt, die Analogien zwischen SQL und Datalog zu veranschaulichen, die syntaktisch normalisierte DB-Sprache NSQL definiert. Normalisierte SQL-Ausdrücke erfüllen drei zusätzliche Anforderungen.

1. Ausschließlich 'Datalog-konforme' SQL3-Operatoren sind anwendbar.
2. Die Schachtelung von Teilausdrücken ist auf die Tiefe eins beschränkt
3. Als Integritätsbedingungen sind nur ASSERTIONs definierbar.

Die Syntax der NSQL-Anfrageausdrücke ist auf SFW-Ausdrücke, den logischen Operatoren AND, NOT (Datalog: $\wedge, \neg$) und den RA-Operator UNION (Datalog: Vereinigungssemantik von Regeln) beschränkt (Restriktion 1). In SQL3 sind beliebig tiefe Schachtelungen von Teilausdrücken (Anfragen, Formeln) zulässig. Um den Aufwand bei der Darstellung sowie bei der später erforderlichen Analyse der SQL-Syntax zu minimieren, werden Schachtelungen von Teilausdrücken teilweise ausgeschlossen (Anfragen in FROM-Klauseln) bzw. auf eine Schachtelungstiefe von eins in NSQL-Formeln beschränkt (Restriktion 2). Beliebig tief geschachtelte Teilausdrücke können mittels Sichten bzw. Hilfssichten (WITH-Option) formuliert werden. Diese Normalisierung erfolgt in Analogie zu der Darstellung von Zwischenergebnissen in Datalog, wo die Formulierung von (Hilfs-)Regeln eine notwendige Konsequenz der syntaktischen Beschränkungen ist. Die spezialisierten SQL-Integritätsbedingungen werden mit Ausnahme der PRIMARY KEYs und und der Fehlerkorrektur mittels NSQL-ASSERTIONs formuliert (Restriktion 3).

An die Definition von NSQL in Abschnitt 3.1 schließen sich in Abschnitt 3.2 Vorschriften zur Transformation von Datalog-Ausdrücken nach NSQL und umgekehrt an.

3.1 NSQL

Im weiteren nicht definierte Begriffe werden in NSQL in gleicher Weise verwendet, wie sie im SQL3-Kontext in Abschnitt 2.2 definiert worden sind. Sie werden auch ohne erneute Definition mit dem Präfix 'NSQL' gekennzeichnet. Aufgrund der in Abschnitt 2.2 ausführlich diskutierten Redundanzen bei SQL-Operatoren und -Ausdrücken, stellen nachfolgende NSQL-Formeln und -Anfragen lediglich eine syntaktische Normalisierung ohne Einschränkungen der Ausdruckskraft dar.

Definition 3.1.1 (NSQL-Formel)

1. *Die aussagelogischen Konstanten* TRUE *und* FALSE *sind NSQL-Formeln.*
2. *Jede atomare SQL-Formel ist eine NSQL-Formel.*
3. *Sind F_1, F_2 NSQL-Formeln und ist F ein NSQL-Atom, dann ist jeder Ausdruck der Form*

   $(F_1 \ \textit{AND} \ F_2)$ $\qquad$ $\textit{NOT} \ (F)$

   eine NSQL-Formel.
4. *Ist Q eine NSQL-Anfrage, dann ist jede existenzquantifizierte NSQL-Anfrage*

   $\textit{EXISTS} \ (Q)$ $\qquad$ $\textit{NOT EXISTS} \ (Q)$

   eine NSQL-Formel.

Definition 3.1.2 (NSQL-Anfrage)

1. *Jeder SQL-*SF*-Ausdruck (Definition 2.2.3) ist eine NSQL-Anfrage.*
2. *Jeder SQL-*SFW*-Ausdruck (Definition 2.2.3) ist eine NSQL-Anfrage, wobei die Formel F in der* WHERE*-Klausel eine NSQL-Formel ist.*
3. *Jeder* UNION*-Ausdruck der Form*

   $Q_1 \ \textit{UNION} \ Q_2$

   ist eine NSQL-Anfrage, wobei Q_1, Q_2 vereinigungskonforme NSQL-Anfragen sind, die nicht vom Typ einer WITH*-Anfrage sind.*
4. *Jeder SQL-*WITH*-Anfrageausdruck (Definition 2.2.3) der Form*

   $\textit{WITH} \ [\ \textit{RECURSIVE} \ ] \ \ S_1, \ldots, S_n \ \ Q$

   ist eine NSQL-Anfrage, wobei Q eine NSQL-Anfrage und $S_1, \ldots, S_n$ NSQL-Sichtspezifikationen sind. Q, $S_1, \ldots, S_n$ verfügen über keine eigenen WITH*-Klauseln mit weiteren Hilfssichten.*

Als Junktoren sind in NSQL-Formeln nur Konjunktion (AND) und Negation (NOT) zugelassen (Datalog: $\wedge$, $\neg$). Der Disjunktionsoperator (OR) entfällt und aufgrund eingeschränkter Schachtelungen kann er auch nicht durch eine negierte Konjunktion ausgedrückt werden. Die Disjunktion wird in NSQL analog zur Vereinigungssemantik mehrerer Regeln in Datalog durch die Vereinigung (UNION) mehrerer Teilanfragen formuliert. Nur atomare Vergleichsformeln und Existenzaussagen können negiert werden. Eine mehrfache Schachtelung negierter NSQL-Formeln ist nicht zulässig. Alle anderen SQL-Mengenoperatoren wie INTERSECT, EXCEPT, etc. lassen sich durch Equi-Joins bzw. NOT EXISTS-Unteranfragen ausdrücken. Das nachfolgende Beispiel zeigt, wie der fehlende Disjunktionsoperator alternativ durch den Basisoperator UNION simuliert werden kann und gibt einen Eindruck, wie die NSQL-Operatoren die Analogien zu Datalog verdeutlichen.

Beispiel 3.1.3 (UNION statt OR-Operator in NSQL)
Seien die beiden Relationen p mit drei und q mit zwei Attributen gegeben.

SQL:
$$\begin{array}{l} \mathit{SELECT}\ \ t_p.a_2,\ t_p.a_1 \quad \mathit{FROM}\ \ p\ \mathit{AS}\ \ t_p,\ \ q\ \mathit{AS}\ \ t_q \\ \mathit{WHERE}\ \ (\ t_p.a_1 = t_q.a_1 \quad \mathit{AND}\ \ t_p.a_3 > 4711\ ) \\ \mathit{OR}\ \ \ \ \ \ (\ t_p.a_3 = t_q.a_2 \quad \mathit{AND}\ \ \mathit{NOT}\,(\ t_p.a_3 < 13\ )\ ) \end{array}$$

NSQL:
$$\begin{array}{l} \mathit{SELECT}\ \ t_p.a_2,\ t_p.a_1 \quad \mathit{FROM}\ \ p\ \mathit{AS}\ \ t_p,\ \ q\ \mathit{AS}\ \ t_q \\ \mathit{WHERE}\ \ (\ t_p.a_1 = t_q.a_1 \quad \mathit{AND}\ \ t_p.a_3 > 4711\ ) \\ \mathit{UNION} \\ \mathit{SELECT}\ \ t_p.a_2,\ t_p.a_1 \quad \mathit{FROM}\ \ p\ \mathit{AS}\ \ t_p,\ \ q\ \mathit{AS}\ \ t_q \\ \mathit{WHERE}\ \ (\ t_p.a_3 = t_q.a_2 \quad \mathit{AND}\ \ \mathit{NOT}\,(\ t_p.a_3 < 13\ )\ ) \end{array}$$

Datalog:
$$\begin{array}{l} \{\ (X,Y)\mid h(X,Y)\ \} \\ \mathit{WITH}\ \ h(Y,X) \leftarrow p(X,Y,Z) \wedge q(X,Z') \wedge Z > 4711 \\ \qquad\quad\ \ h(Y,X) \leftarrow p(X,Y,Z) \wedge q(Z',Z) \wedge \neg\ Z < 13 \end{array}$$

Die Begrenzung der Schachtelungstiefe für Unteranfragen auf die maximale Stufe von eins erzwingt die Strukturierung unüberschaubarer SQL-Anfragen in gut lesbare NSQL-Anfragen mit mehreren ebenfalls gut strukturierten Hilfssichten. Die Struktur einer NSQL-Anfrage/Sicht ist somit genauso flach wie die einer Datalog-Anfrage/Regel. Damit wird das Konzept der Hilfssichten in NSQL genauso wie bei Datalog erforderlich, um komplexe Anfragen formulieren zu können. Daß eine solche Normalisierung ohne Einschränkungen der Ausdruckskraft möglich ist, zeigt u.a. Date in [DD97] 13.1, wo die Auswertung von Sichten, die wiederum Sichten referenzieren, anhand des inversen Prozesses erläutert wird. Cremers/Griefahn/Hinze zeigen in [CGH94] 3.2 ein Verfahren, mittels dem prädikatenlogische Programme nach Datalog-Regeln ohne Einschränkung der Semantik transformiert werden können. Bei [Schoe95] 2.2 werden Normalisierungsverfahren für TRC- und DRC-basierte Sprachen vorgestellt.

Eine SQL-Sicht mit einem Anfrageausdruck Q ist eine **NSQL-Sicht**, gdw. Q eine NSQL-Anfrage ist. Eine SQL-ASSERTION mit einer Formel F als Bedingung ist eine **NSQL-ASSERTION**, gdw. F eine NSQL-Formel ist. Hinsichtlich der Transformationsfunktionen sei darauf hingewiesen, daß ein Anfrage-Ausdruck Q in F ggf. eine WITH-Klausel enthalten kann, und daß in der FROM-Klausel von Q nur ein Relationssymbol auftritt (Abschnitt 2.2.3). Daß Basisrelationen-Constraints (PRIMARY, UNIQUE, FOREIGN KEY und CHECK) äquivalent als Formel einer ASSERTION definiert werden können, wurde bereits für die SQL-Integritätsbedingungen erörtert. Dort wurde ebenfalls auf die Redundanz von Domänen-Constraints bzgl. Attribut-Constraints hingewiesen, die wiederum redundant zu Basisrelationen-Constraints sind. Eine Beschränkung auf ASSERTIONs hat somit nur den Verlust der fehlerkorrigierenden Reaktionen des FOREIGN KEY-Constraints und den Verlust von Primärschlüsseln zur Folge. Die beiden SQL3-Integritätsprüfungszeitpunkte IMMEDIATE und DEFERRED gelten in NSQL unverändert.

Die für Basis-SQL in Abschnitt 2.2.3 definierten Begriffe einer **SQL-Datenbank** $\mathcal{D}^{\mathcal{S}}$ (Definition 2.2.11), einer **Signatur** Σ^{S} (Definition 2.2.10) und eines **DB-Schemas** $\mathcal{S}^{\mathcal{S}}$ (Definition 2.2.9) gelten auch im NSQL-Kontext, sofern $\mathcal{RV}$ eine Menge virtueller NSQL-Sichten ist und $\mathcal{C}$ eine Menge von NSQL-ASSERTIONs. $\mathcal{RM}$ ist auch in NSQL eine leere

Menge materialisierter Sichten und $\mathcal{T}$ eine Menge eine Menge von SQL3-Triggern.

Die aus Datalog bekannten Definitionen eines **Abhängigkeitsgraphen** für Relationen und Integritätsbedingungen (2.1.12) und eines Abhängigkeitsgraphen für Attribute (2.1.13) können analog in NSQL angewendet werden. Nur der Begriff des positiven und negativen Auftretens von Relationssymbolen in SQL-Ausdrücken muß an die SQL-Syntax angepaßt werden. Aufgrund der Normalisierung von NSQL sind die Analogien zu den Begriffen positiver und negativer Rumpfliterale offenkundig. Sei ($S \in \mathcal{R} \cup \mathcal{C}$) die Definition einer Sicht oder einer Integritätsbedingung von $\mathcal{D}^{\mathcal{S}}$ und $R \in rel(\mathcal{R} \cup \mathcal{B})$ sei das Symbol einer Relation.

- R tritt positiv in S auf, gdw. R in der FROM-Klausel der Anfrage Q auftritt oder in der FROM-Klausel einer EXISTS-quantifizierten Unteranfrage (Datalog: positives Literal).
- R tritt negativ in S auf, gdw. R in der FROM-Klausel einer NOT EXISTS-quantifizierten Unteranfrage auftritt (Datalog: negatives Literal).

Ausgehend von diesem Verständnis der Polarität des Auftretens eines Relationssymbols in einer Sicht oder Integritätsbedingung kann die Definition der Abhängigkeitsgraphen unmittelbar in SQL angewendet werden. Der Abhängigkeitsgraph ist vergleichbar zu dem im SQL-Standard ([ANSI99II] 7.12: query name dependency graph). Jedoch fehlt im SQL-Standard die für Propagierungsverfahren sehr wichtige Information der Polarität des Auftretens.

Da Datalog (anders als in der Literatur üblich) in einer um eine WITH-Klausel mit Hilfssichten erweiterten Form eingeführt worden ist, sind die wesentlichen Erweiterungen, die für eine Anwendung des Begriffs der **Abhängigkeiten** (Definition 2.1.14) auch in SQL notwendig sind, bereits in Abschnitt 2.1.1 durchgeführt worden. Aufgrund der Normalisierung von NSQL können somit die Informationen der Abhängigkeitsgraphen analog zu dem Vorgehen in Datalog unmittelbar auch für die **Stratifikation** (Definition 2.1.15) von Sichtmengen in SQL und die **Klassifizierung** (Definition 2.1.16) von SQL-Sichtmengen und Datenbanken verwendet werden.

Die Beschränkungen der **NSQL-Änderungssprache** gegenüber SQL3 ergeben sich ausschließlich aus der Verwendung von NSQL-Anfragen und -Formeln. Das SQL3-Transaktionskonzept wird unverändert nach NSQL übernommen.

Abschließend werden NSQL-Anweisungen zur für das DB-Schema des Beispiels Beispiel 2.1.17 veranschaulicht (Abb. 2.1). Da weder in NSQL noch in SQL3 materialisierte Sichten definierbar sind, werden alle Sichten als virtuelle Sichten definiert.

Beispiel 3.1.4 (Unternehmensorganisation)
Die beiden Basisrelationen werden mittels folgender CREATE-Befehle erzeugt.

CREATE BASERELATION mitarbeiter (mita, abteil, gehalt, mgr);
CREATE BASERELATION abteilung (nr, bez);.

Zur Erzeugung der drei Sichten werden folgende CREATE VIEW-Befehle spezifiziert.

- *Mitarbeiter-Informationen und das Attribut des Abteilungskürzels:*

  CREATE VIEW $mita\_abt\ (mita, nr, bez, mgr)$ *AS*
  SELECT $t_m.mita,\ t_m.abteil,\ t_a.bez,\ t_m.mgr$
  FROM $mitarbeiter$ *AS* $t_m,\ abteilung$ *AS* t_a
  WHERE $t_m.abteil = t_a.nr;$

- *Mitarbeiter mit ihren (in-)direkten Managern:*

  CREATE VIEW $mita\_mgr\ (mita,\ bez,\ mgr)$ *AS WITH RECURSIVE*
  $h_1(mita, bez, mgr)$ *AS SELECT* $t_{h_1}.mita,\ t_{h_1}.bez,\ t_{ma}.mgr$
  FROM h_1 *AS* $t_{h_1},\ mita\_abt$ *AS* t_{ma}
  WHERE $t_{h_1}.mgr = t_{ma}.mita$
  UNION SELECT $t_{ma}.mita,\ t_{ma}.bez,\ t_{ma}.mgr$
  FROM $mita\_abt$ *AS* $t_{ma},$
  SELECT $t_{h_1}.mita, t_{h_1}.bez, t_{h_1}.mgr$ *FROM* h_1 *AS* $t_{h_1};$

- *Mitarbeiter, mit ihren Managern, die nicht der Abteilung 'Management' (Nr. 13) angehören:*

  CREATE VIEW $lower\_mgr\ (mita,\ bez,\ mgr)$ *AS*
  SELECT $t_{mm}.mita,\ t_{mm}.bez,\ t_{mm}.mgr$
  FROM $mita\_mgr$ *AS* t_{mm}
  WHERE NOT EXISTS (*SELECT* 1 *FROM* $mitarbeiter$ *AS* t_m
  WHERE $t_m.abteil = 13$ *AND* $t_m.mita = t_{mm}.mgr$).

Für die beiden Integritätsbedingungen werden folgende ASSERTIONs definiert:

- *Inklusionsbedingung des Attributs mitarbeiter.abteil zu abteilung.nr:*

  CREATE ASSERTION ic_1 *CHECK* (*NOT EXISTS* (
  SELECT 1 FROM $mitarbeiter$ *AS* t_m *WHERE NOT EXISTS* (
  SELECT 1 FROM $abteilung$ *AS* t_a
  WHERE $t_a.nr = t_m.abteil$))) *DEFERRED;*

- *Manager müssen der gleichen Abteilung wie die Mitarbeiter angehören oder dem 'Management' ('MG').*

  CREATE ASSERTION ic_2 *CHECK* (*NOT EXISTS* (
  SELECT 1 FROM $mita\_mgr$ *AS* t_{mm} *WHERE NOT EXISTS* (
  SELECT 1 FROM $mita\_abt$ *AS* t_{ma}
  WHERE $t_{ma}.bez = t_{mm}.bez$ *AND* $t_{ma}.mita = t_{mm}.mgr$
  UNION
  SELECT 1 FROM $mita\_abt$ *AS* t_{ma}
  WHERE $t_{ma}.bez =$'MG' *AND* $t_{ma}.mita = t_{mm}.mgr$))) *IMMEDIATE.*

3.2 Transformationsvorschriften

Ziel dieses Abschnitts ist es, grundlegende Gemeinsamkeiten und Unterschiede zwischen Datalog und NSQL aufzuzeigen. In den Abschnitten 3.2.1 und 3.2.2) werden Transformationsvorschriften definiert, mit deren Hilfe Datalog-Anfragen/Regeln/Integritätsbedingungen auf NSQL-Ausdrücke und umgekehrt abgebildet werden können. In Abschnitt 2.2

wurde bereits darauf hingewiesen, daß auf dem Weg der Transformation einer NSQL-Anfrage/Regel/Integritätsbedingung in einen semantisch äquivalenten Datalog-Ausdruck deren Semantik spezifiziert wird.

Beiträge, die die semantische Äquivalenz zwischen verschiedenen in der Wissenschaft verwendeten Sprachen wie sicherem Datalog, sicherem TRC und sicherem DRC, sowie ihre relationale Vollständigkeit zeigen, sind in der Literatur vielfach zu finden ([Ull89], [CGT90], [Cre92], [KK93], [CGH94]). In der Regel basiert der Äquivalenznachweis auf der syntaktischen Transformation der Ausdrücke der verschiedenen Sprachen ineinander. Die Transformation von SQL in eine andere formale Sprache wird seltener durchgeführt. Ein Grund ist sicherlich die Komplexität der SQL-Syntax. Ullman z.B. führt in [Ull89] den Nachweis, daß die fünf RA-Operatoren mittels SELECT-Ausdrücken formuliert werden können. Auf diese Transformation von SQL-Anfragen in RA-Ausdrücke wird sehr häufig im Rahmen der Anfrageoptimierung zurückgegriffen. Die Überführbarkeit einer relevanten Teilmenge von SQL1-Anfragen (SELECT-Ausdrücke) in RA-Terme haben Ceri et.al. in [CG85], [CGT90] gezeigt. Negri et.al. berücksichtigen in [NPS91], daß SQL eine dreiwertige Logik zugrunde liegt, und überführen SELECT-Ausdrücke über den Zwischenschritt eines dreiwertigen Kalküls in Ausdrücke eines zweiwertigen Kalküls. Daß sich diese Beiträge auf SQL1 beziehen, beeinträchtigt ihre Aktualität nur wenig. Gegenüber SQL3 (NSQL) fehlen ihnen jedoch die Hilfssichten der WITH-Klausel und rekursive Sichten und Anfragen. Über den Rahmen dieser Arbeit gehen sie insofern hinaus, als daß sie Aggregatfunktionen (SUM, MIN, MAX, AVG,...), Gruppierungen (GROUP BY, HAVING) und eine Vielzahl der Vergleichsoperatoren (IN, SOME, ANY, ALL,...) berücksichtigen.

Problematisch in Hinblick auf das Ziel dieses Abschnitts ist insbesondere, daß die bisherigen Beiträge nicht die unmittelbare Transformation nach Datalog zeigen. Anders als in [CG85], [NPS91], wo Transformationsregeln verschiedene 'Basistypen' komplexer SQL-Anfragen (IN, [NOT] EXISTS, ALL-, Aggregat-Anfragen) in ihrer Gesamtheit überführen, werden hier im weiteren beliebige sichere NSQL- und Datalog-Ausdrücke in ihre syntaktischen Komponenten zerlegt und in die semantisch äquivalenten Komponenten der anderen Sprache transformiert. Dieses teilausdrucksbezogene Transformationsvorschriften haben Ceri/Gottlob/Tanca in [CGT90] nur informell und nicht vollständig spezifiziert und ihre Idee lediglich anhand einiger Beispiele verdeutlicht.

Da NSQL viele Probleme, die aus der syntaktischen Komplexität resultieren, vermeidet, ist NSQL im Gegensatz zu SQL besser für eine anschauliche Darstellung der Transformation nach Datalog geeignet. Da NSQL syntaktisch wie semantisch eine echte Teilmenge von SQL3 darstellt, können alle sicheren Datalog-Anfragen und -Regeln ohne Einschränkung nach SQL3 transformiert werden. Der Nachweis der umgekehrten Aussage, daß SQL3-Anfragen und -Regeln vollständig in sichere Datalog-Ausdrücke überführt werden können, wird in dieser Arbeit aus zwei Gründen nicht geführt. Zum einen muß Datalog um benutzerdefinierte Funktionen[1] erweitert werden, damit alle SQL-Ausdrücke nach Datalog transformieren werden können, was jedoch nicht Thema dieser Arbeit ist. Zum anderen ist aufgrund der vielen syntaktisch unterschiedlichen, aber semantisch äquivalenten Teilausdrücke in SQL3 (Abschnitt 2.2), die Spezifikation von SQL3-Transformationsregeln

[1] Datalog selbst ist eine Sprache frei von Funktionen. Die Problematik einer Erweiterung um Funktionsausdrücke (Datalog^{fun}) wird u.a. in [AH88], [Ull89], [CGH94] diskutiert.

ebenso aufwendig wie auch trivial vollständig zu definieren.

Unterschiede:

Ein hinsichtlich seiner Konsequenzen sehr weitreichender Unterschied ist, daß Datalog als kalkülreine Sprache auf dem DRC basiert und SQL als Hybridsprache sowohl auf dem TRC wie auch auf der RA. Der SELECT-Ausdruck als das zentrale Konstrukt zur Formulierung einer Anfrage basiert auf dem TRC, wobei mehrere SELECT-Ausdrücke mittels RA-Operatoren (UNION, JOIN,...) verknüpft werden können. Alternativ sind SELECT-Ausdrücke auch als RA-basierte Join-Ausdrücke formulierbar. Ein Problem stellt in diesem Zusammenhang die unterschiedlichen Variablennotationen und ihre Transformation dar. In Datalog wird eine DRC-basierte, positionelle Schreibweise der Attribute (Bereichsvariablen) verwendet, in SQL eine TRC-basierten Notation mit Tupelvariablen und Attributsymbolen. Wie diese Unterschiede aufgehoben werden können, hat u.a. Ullman bereits in [Ull89] anhand eines Algorithmus zur Transformation eines TRC-Ausdrucks in einen DRC-Ausdruck spezifiziert. Dabei ersetzt Ullmann das Auftreten von Komponenten der Tupelvariablen durch entsprechend zu generierende Bereichsvariablen. Dieser Ansatz findet auch bei den hier definierten Transformationsvorschriften Anwendung.

Aus den verschiedenen Kalkülen resultiert ferner die unterschiedliche Darstellung von Identitätsvergleichen. In Datalog werden sie i.d.R. durch die Verwendung der gleichen Bereichsvariablen in den verschiedenen Rumpfliteralen formuliert (z.B.: $p(X,Y) \wedge q(Y,Z)$). In SQL werden sie ausschließlich als atomare Vergleichsformeln in der WHERE-Klausel dargestellt (z.B.: $tv_p.a_2 = tv_q.a_1$).

Die Problematik der impliziten Existenzquantifizierung zu Beginn des Regelrumpfs in Datalog und der daraus resultierenden Forderung nach Bereichsbeschränkung sind bereits in Abschnitt 2.1 erörtert worden. Da jedoch nur bereichsbeschränkte Datalog-Regeln/Anfragen transformiert werden, ist es unproblematisch, den impliziten (negierten) Existenzquantor durch einen expliziten (NOT) EXISTS-Operator in NSQL auszudrücken.

Ein weiterer wesentlicher Unterschied ist die unterschiedliche Terminologie beider Sprachen. So werden analoge Konzepte mit unterschiedlichen Fachbegriffen benannt, wie z.B. bei Deduktionsregeln und Sichten. Probleme werden zudem auf verschiedene Weisen diskutiert. Ein Beispiel ist die im Datalog-Kontext sehr ausführlich geführte Diskussion der Bereichsbeschränkung von Anfragen und Regeln mit ihrer Bedeutung für andere Fragestellungen. Im Kontext der anwendungsorientierten Sprache SQL tritt dieses Problem scheinbar weder in den ANSI-Publikationen noch in der Sekundärliteratur auf. Tatsächlich sind jedoch in SQL Anfragen (ohne Angabe von Gründen) syntaktisch derart eingeschränkt, daß nur 'bereichsbeschränkte' Anfragen formulierbar sind.

Analogien:

Die beiden DB-Sprachen sind in den vorangegangen Abschnitten detailliert erörtert worden, so daß eine ausführliche Motivation der Analogien hier nicht mehr erforderlich ist.

- Sichere Datalog-Anfragen, -Regeln und -Integritätsbedingungen können in semantisch äquivalente NSQL-Anfragen, -Sichten und -ASSERTIONs transformiert werden und umgekehrt. Mit SQL3 können nun auch rekursive Datalog-Anfragen und Regeln

nach SQL transformiert werden. Jede SQL-Anfrage (auch mit UNION-Operator und Hilfsregeln) kann aufgrund der Verwendung von Hilfsregeln im Datalog-Kontext in genau eine Datalog-Anfrage überführt werden. Lediglich die argumentlosen Datalog-Anfragen und -Relationen müssen in SQL simuliert werden.

- Die Konzepte der Hilfsregeln entsprechen einander in beiden Sprachen.

Die Entsprechungen zwischen beiden Sprachen ergeben sich unmittelbar aus den Transformationsvorschriften in den nachfolgenden Abschnitten. Eine präzise Erörterung der Analogien würde die Inhalte der Funktionsdefinitionen vorwegnehmen und eine weniger präzise Diskussion würde den Schwierigkeiten der Transformationsschritte nicht gerecht werden, so daß an dieser Stelle auf eine allgemeine Diskussion verzichtet wird.

Um die Strukturen der komplexen Mengen der Γ- und Ψ-Funktionen zu veranschaulichen, aber, auch um ggf. Wiederholungen in verschiedenen Definitionen zu vermeiden, wird in einer Definition eine Gruppe von Transformationsfunktionen zusammengefaßt, die gemeinsam eine bestimmte Übersetzungsaufgabe lösen. Um Erklärungen trotzdem unmittelbar bei einer Funktionsdefinition aufführen zu können, enthalten die Definitionen ggf. Anmerkungen, die über den üblichen Rahmen einer Definition hinausgehen.

3.2.1 Transformation Datalog → NSQL

Ausgehend von einem Datalog-DB-Schema $\mathcal{S}^{\mathcal{D}}$ werden die Spezifikationen der Basis- und abgeleiteten Relationen, der Deduktionsregeln und Integritätsbedingungen mittels der Γ-Funktionen in NSQL-DDL-Befehle transformiert. Mittels dieser generierten NSQL-DML-Befehle kann dann ein äquivalentes NSQL-DB-Schema S^S erzeugt werden. Die Gesamtaufgabe der Transformation ist aufgrund der Komplexität in verschiedene Einzelfunktionen aufgeteilt, deren Ergebnisse (Zeichenketten) miteinander konkateniert werden (binärer Konkatenationsoperator: ||). Die drei Punkte (...) symbolisieren den wiederholten Aufruf von Funktionen und die Konkatenation ihrer Ergebnisse gemäß der Anzahl der zu transformierenden (Teil-)Ausdrücke. Die beiden Hochkomma (' ') bezeichnet die leere Zeichenkette. Struktur und Bezeichnung der Funktionen orientiert sich daran, welchen NSQL-Teilausdruck sie als Ergebnis liefern. Die Aufrufe der verschiedenen Strukturen wird in der Abbildung 3.1 skizziert.

- Mit den Funktionen $\mathbf{\Gamma_{CreateR}}, \mathbf{\Gamma_{CreateC}}, \mathbf{\Gamma_{Q}}$ wird eine sichere Datalog-Regel (R), Integritätsbedingung (C) oder Anfrage (Q) in einen syntaktisch korrekten NSQL-Befehl zur Erzeugung einer Sicht oder ASSERTION bzw. in eine NSQL-Anfrage transformiert. Diese Prozeduren können von einem Anwender zur Übersetzung einer einzelnen Datalog-Regel, Integritätsbedingung oder Anfrage aufgerufen werden, oder sie werden von einem Transformationsprogramm wiederholt zur Übersetzung der DB-Objekte eines Datalog-DB-Schemas aufgerufen. Für die Transformation werden den Funktionen $\Gamma_{CreateR}, \Gamma_{CreateC}$ das Relationssymbol der Regel bzw. Integritätsbedingung übergeben, der Γ_Q-Funktion die Spezifikation der Anfrage.

- Die **Γ-Regelfunktionen** $\Gamma_{CreateR}$, Γ_R, Γ_{RHead}, Γ_U bilden eine Datalog-Regel auf eine NSQL-Sicht ab. $\Gamma_{CreateR}$ überführt die Definitionen von Regeln mit dem gleichen Relationssymbol im Kopfliteral in eine CREATE VIEW-Anweisung und Γ_R die Regeldefinitionen in die Definition einer Sicht. Γ_{RHead} bildet ein Kopfliteral auf das

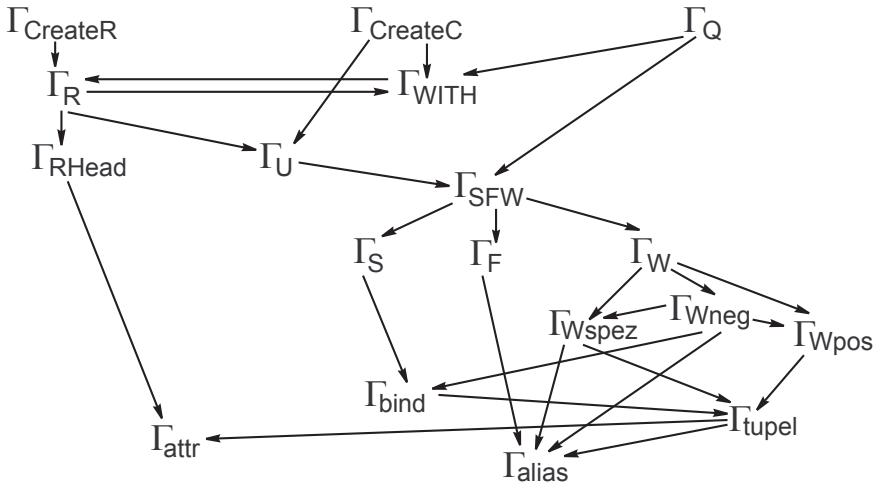

Abbildung 3.1: Transformationsfunktionen Datalog $\longrightarrow$ NSQL

Literal der Sicht ab. Γ_U transformiert eine Menge von Regeln mit dem gleichen Relationssymbol im Kopfliteral in SFW-Ausdrücke, die mittels UNION-Operatoren verknüpft werden.

- Die **Γ-Transformationsfunktionen** Γ_{SFW}, Γ_S, Γ_F, Γ_W, Γ_{Wpos}, Γ_{Wneg}, Γ_{Wspez}, Γ_{WITH} transformieren eine Regeldefinition in einen SFW-Ausdruck, wobei Γ_S, Γ_F und Γ_W die SELECT-, FROM- und WHERE-Klausel generieren. Die Funktionen Γ_{Wpos}, Γ_{Wneg} und Γ_{Wspez} bilden Datalog-Rumpfliterale (positive/negative DB-Atome und Vergleichsatome) auf Teilformeln der WHERE-Bedingung ab.

- Die **Γ-Namensfunktionen** Γ_{alias}, Γ_{attr}, Γ_{tupel}, Γ_{bind} sind Hilfsfunktionen zur Abbildung von Bereichsvariablen auf NSQL-Attributterme. Γ_{alias} bildet Datalog-Relationssymbole auf Tupelvariablen ab. Γ_{attr} ordnet einer Argumentposition ein Attributsymbol zu. Γ_{tupel} bildet eine Bereichsvariable auf einen Attributterm ab und Γ_{bind} einen Datalog-Term auf einen NSQL-Term. Die Funktionen gewährleisten die Eindeutigkeit der generierten Symbole in einem Kontext (z.B. Formel), der ebenfalls als Eingabeparameter übergeben wird.

- Die Funktionen Γ_F, Γ_{Wpos} und Γ_{Wspez} rufen sogenannte **Stern-Funktionen** Γ_{F^*}, Γ_{Wpos^*} und Γ_{Wspez^*} auf, die aus Gründen der Übersichtlichkeit nicht in der Abbildung aufgeführt sind. Die Transformation selbst wird von den Stern-Funktionen ausgeführt, deren Ergebnisse in den aufrufenden Funktionen konkateniert werden.

Im weiteren sei ein Datalog-Schema $\mathcal{S}^{\mathcal{D}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C} \rangle$ gegeben mit der Signatur $\Sigma^D = \langle \mathcal{PS}, \mathcal{AS} \rangle$. Für alle Funktionen ist das Schema $\mathcal{S}^{\mathcal{D}}$ impliziter Eingabeparameter, der jedoch aus Gründen der Übersichtlichkeit nicht aufgeführt wird.

Definition 3.2.1 (Γ-Namensfunktionen)
Sei $B := L_1 \wedge ... \wedge L_n$ eine Datalog-Formel und $rel(L_i)$ ein Prädikatsymbol des DB-Schemas $\mathcal{S}^{\mathcal{D}}$ ($rel(L_i) \in \mathcal{PS}$, $1 \leq i \leq n$).

- *Γ_{alias} bildet das Relationssymbol eines Literals L_i aus einer gegebenen Formel B auf einen Aliasnamen ab, der als Symbol einer Tupelvariable in NSQL verwendet wird.*

$$\Gamma_{alias}(B, L_i) := \begin{cases} rel(L_i) & \text{falls } rel(L_i) \text{ nur einmal in } B \text{ auftritt} \\ rel(L_i) \| \text{'\_'} \| k & \text{falls } rel(L_i) \text{ k-mal in } B \text{ auftritt } (1 < k \leq n). \end{cases}$$

- *Γ_{attr} bildet eine Attributposition j eines gegebenen Literals $L(t_1, ..., t_m)$ mit $1 \leq j \leq m$ auf ein NSQL-Attributsymbol der NSQL-Relation $rel(L)$ ab.*

$$\Gamma_{attr}(L, j) := \text{'attr\_name\_'} \| rel(L) \| \text{'\_'} \| j.$$

- *Γ_{tupel} bildet eine Attributposition j eines gegebenen Literals $L_i(t_1, ..., t_m) \in B$ auf einen NSQL-Attributterm ab ($1 \leq j \leq m$).*

$$\Gamma_{tupel}(B, L_i, j) := \Gamma_{alias}(B, L_i) \| \text{'.'} \| \Gamma_{attr}(L_i, j).$$

- *Γ_{bind} bildet einen Term t_j eines gegebenen Literals $L_i(t_1, ..., t_m) \in B$ mit $1 \leq j \leq m$ auf sich selbst ab, wenn t_j eine Konstante ist. Ist t_j eine Bereichsvariable, die zudem an der Argumentposition g eines positiv in B auftretenden Literals L_h auftritt, dann wird t_j auf das Symbol des g-ten Attributs der Relation $rel(L_h)$ abgebildet.*

$$\Gamma_{bind}(B, t_j) := \begin{cases} \Gamma_{tupel}(B, L_h, g) & \text{falls } t_j \in vars(L_h) \text{ und } L_h \text{ ist positiv in } B \\ t_j & \text{falls } t_j \in const(B). \end{cases}$$

Die Γ-Transformationsfunktionen bilden die positiven und negativen Literale von Datalog-Formeln auf NSQL-FROM- und WHERE-Klauseln ab. Die Zielliste einer Anfrage bzw. das Kopfliteral einer Regel wird auf die SELECT-Klausel abgebildet.

Definition 3.2.2 (Γ-Transformationsfunktionen)
Sei eine Datalog-Formel $B := L_1 \wedge \ldots \wedge L_k \wedge N_1 \wedge \ldots \wedge N_m$ als eine Konjunktion positiver (L_i, $1 \leq i \leq k$) und negativer (N_j, $0 \leq j \leq m$) Literale gegeben. B sei der Qualifikationsteil einer gegebenen Anfrage/Regel/Integritätsbedingung und $t_1, ..., t_n$ seien die Parameter der Zielliste bzw. des Kopfliterals mit $n \geq 0$.

- *Γ_S bildet die Liste der Parameter $[t_1, \ldots, t_n]$ auf eine NSQL-SELECT-Klausel ab. Für die Simulation parameterloser Anfragen/Regeln ($n = 0$) in NSQL wird ein konstanter Wert selektiert.*

$$\Gamma_S(B, [t_1, ..., t_n]) := \begin{cases} \text{'SELECT'} \parallel \Gamma_{bind}(B, t_1) \parallel \text{','} \parallel \ldots \parallel \text{','} \parallel \Gamma_{bind}(B, t_n) & \text{falls } n > 0 \\ \text{'SELECT 1 '} & \text{falls } n = 0. \end{cases}$$

- Γ_F *bildet die Liste der in einer gegebenen Formel B positiv auftretenden Literale* $[L_1, ..., L_k]$*, die DB-Atome sind, auf eine NSQL-FROM-Klausel ab.*

$$\Gamma_F(B, [L_1, ..., L_k]) \ := \ \text{'FROM'} \parallel \Gamma_{F^*}(B, L_1) \parallel \text{','} \parallel \ldots \parallel \text{','} \parallel \Gamma_{F^*}(B, L_k).$$

  - *Sei L ein in einer gegebenen Formel B positiv auftretendes DB-Atom, dann bildet* Γ_{F^*} *L auf einen NSQL-Relationssymbol und eine Tupelvariable ab.*

$$\Gamma_{F^*}(B, L) \ := \ rel(L) \parallel \text{'AS'} \parallel \Gamma_{alias}(B, L).$$

- Γ_W *bildet die in einer gegebenen Formel B auftretenden gemeinsamen Variablen positiver Literale, die Vergleichsatome, die Konstanten und die negativen Literale auf eine NSQL-WHERE-Klausel ab* ($\dagger \in \{_{Wpos}, _{Wspez}, _{Wneg}\}$).

$$\Gamma_W(B) \ := \ \begin{cases} \text{'WHERE TRUE'} & \exists \Gamma_\dagger \mid \Gamma_\dagger \ \neq \ \text{' '} \\ \parallel \Gamma_{Wpos}(B, L_1, L_2) \parallel ... \parallel \Gamma_{Wpos}(B, L_1, L_k) \parallel ... & \\ \parallel \Gamma_{Wpos}(B, L_k, L_1) \parallel ... \parallel \Gamma_{Wpos}(B, L_{k-1}, L_k) & \\ \parallel \Gamma_{Wspez}(B, L_1) \parallel ... \parallel \Gamma_{Wspez}(B, L_k) & \\ \parallel \Gamma_{Wneg}(B, N_1) \parallel ... \parallel \Gamma_{Wneg}(B, N_m) & \\ & \\ \text{' '} & sonst \end{cases}$$

  - Γ_{Wpos} *bildet alle gemeinsamen Bereichsvariablen zweier gegebener DB-Atome* $L_g(t_{g1}, ...t_{gn'})$, $L_h(t_{h1}, ...t_{hn''})$*, die positiv in B auftreten, als Konjunktion von Identitätsvergleichen der zugehörigen NSQL-Tupelkomponenten ab.*

$$\begin{aligned} \Gamma_{Wpos}(B, L_g, L_h) \ := \ & \Gamma_{Wpos^*}(B, L_g, L_h, t_{g1}, t_{h1}) \parallel \Gamma_{Wpos^*}(B, L_g, L_h, t_{g1}, t_{h2}) \\ & \parallel ... \parallel \Gamma_{Wpos^*}(B, L_g, L_h, t_{gn'}, t_{hn''}). \end{aligned}$$

  - *Seien* L_g, L_h *zwei Literale, die positiv in einer gegebenen Formel B auftreten, dann bildet* Γ_{Wpos^*} *eine Bereichsvariable, die als Terme* $t_{gi'}, t_{hi''}$ *in beiden Literalen auftritt* ($t_{gi'}, t_{hi''} \in vars(L_g \cup L_h) \ \wedge \ t_{gi'} = t_{hi''}$) *als Identitätsvergleich der zugehörigen NSQL-Tupelkomponenten ab.*

$$\Gamma_{Wpos^*}(B, L_g, L_h, t_{gi'}, t_{hi''}) \ := \ \text{'AND'} \parallel \Gamma_{tupel}(B, L_g, i') \parallel \text{'='} \parallel \Gamma_{tupel}(B, L_h, i'').$$

  - Γ_{Wspez} *bildet ein Literal L aus einer gegebenen Formel B auf einen NSQL-Vergleichsausdruck ab, wenn L ein Vergleichsliteral ist. Ist L ein DB-Atom, dann werden die Parameter von L auf Konjunktionen von NSQL-Identitätsvergleichen abgebildet.*

$$\Gamma_{Wspez}(B, L) \ := \ \begin{cases} \Gamma_{Wspez^*}(B, L, t_1, t_2) \parallel & falls \ L = L(t_1, ..., t_n) \\ \Gamma_{Wspez^*}(B, L, t_1, t_3) \parallel \ ... \ \parallel & \\ \Gamma_{Wspez^*}(B, L, t_{(n-1)}, t_n) & \\ & \\ \text{'AND'} \parallel \Gamma_{bind}(B, t') \parallel \Theta \parallel \Gamma_{bind}(B, t'') & falls \ L_i = t' \Theta t''. \end{cases}$$

– *Sei L ein DB-Atom einer gegebenen Formel B und seien $t_{j'}, t_{j''}$ zwei Parameter von L, die mit der gleichen Bereichsvariablen belegt sind, dann bildet Γ_{Wspez^*} die beiden Terme auf einen NSQL-Identitätsvergleich ab. Ist $t_{j'}$ eine Konstante, dann liefert Γ_{Wspez^*} als Ergebnis einen Identitätsvergleich von $t_{j'}$ mit der zugehörigen NSQL-Tupelkomponente.*[2]

$$\Gamma_{Wspez^*}(B, L, t_{j'}, t_{j''}) \;:=\; \begin{cases} \text{'AND'} \| \Gamma_{tupel}(B, L, j') \| & \text{falls } t_{j'} = t_{j''} \wedge \\ \text{'='} \| \Gamma_{tupel}(B, L, j'') & t_{j'}, t_{j''} \in vars(L) \\ & \\ \text{'AND'} \| \Gamma_{tupel}(B, L, j') \| \text{'='} \| t_{j'} & \text{falls } t_{j'} \in const(L). \end{cases}$$

– *Sei N ein in B negativ auftretendes Literal. Ist N ein negatives Vergleichsliteral, so wird N von Γ_{Wneg} auf eine negierte NSQL-Vergleichsformel abgebildet. Ist N ein negatives DB-Atom, so bildet Γ_{Wneg} N auf eine NSQL-NOT EXISTS-Formel ab. Die Bindungen der Variablen, die als Parameter von N wie auch in positiven Literalen $L_1, ..., L_k \in B$ auftreten werden durch Γ_{Wpos} jeweils auf eine NSQL-Vergleichsformel abgebildet.*[3]

$\Gamma_{Wneg}(B, N) \;:=$

$$\begin{cases} \text{'AND NOT ('} \| & \text{falls } N = \neg t_{j'} \Theta t_{j''} \\ \Gamma_{bind}(B, t_{j'}) \| \Theta \| \Gamma_{bind}(B, t_{j''}) \| \text{')'} & \\ & \\ \text{'AND NOT EXISTS (SELECT 1} & \text{falls } N = \neg N(t_1, .., t_n). \\ \text{FROM'} \parallel rel(N) \parallel \text{'AS'} \parallel \Gamma_{alias}(B, N) \| & \\ \text{'WHERE TRUE'} \| & \\ \Gamma_{Wpos}(B, N, L_1) \| \ldots \| \Gamma_{Wpos}(B, N, L_k) & \\ \| \Gamma_{Wspez}(B, N) \| \text{')'} & \end{cases}$$

- Γ_{SFW} *bildet eine gegebene Formel B mit den positiven DB-Atomen $L_1, ..., L_k$ und eine gegebene Liste von Termen $[t_1, ..., t_n]$ auf einen NSQL-SFW-Ausdruck ab.*

$$\Gamma_{SFW}(B, [t_1, ..., t_n]) \;:=\; \Gamma_S(B, [t_1, ..., t_n]) \parallel \Gamma_F(B, [L_1, ..., L_k]) \parallel \Gamma_W(B).$$

- $H_1^{help}, ..., H_{l'}^{help}$ *sei eine gegebene Liste von Relationssymbolen von Hilfsregeln. Γ_{WITH} bildet auf der Basis des gegebenen Schemas $\mathcal{S}^{\mathcal{D}}$ die Hilfsregeln $R_1^{help}, ..., R_l^{help}$ $(1 \leq l' \leq l)$ auf eine NSQL-WITH-Klausel ab. (Γ_R: Definition 3.2.3)*

[2] Aus Gründen einer vereinfachten Darstellung wird die Generierung redundanter NSQL-Teilformeln akzeptiert. So wird z.B. für ein Literal $p(a, X, Y)$ Γ_{Wspez^*} jeweils für die Terme a, X und a, Y aufgerufen, so daß die NSQL-Teilformel 'AND $rel\_name.attr_1$ ='a" zweimal generiert wird. Bei einer Implementierung der Funktionen sind solche Redundanzen zu vermeiden.

[3] Auch hier wird mit dem Ziel einer gut lesbaren Darstellung die Generierung redundanter NSQL-Teilformeln akzeptiert, die für Variablen von N erzeugt werden, die mehrfach in positiven Literalen gebunden werden.

$$\Gamma_{WITH}([H_1^{help}, ..., H_{l'}^{help}]) :=$$

$$\begin{cases} \text{'WITH'} \parallel & \text{falls } R_1^{help}, ..., R_l^{help} \text{ nicht} \\ \Gamma_R(H_1^{help}) \parallel \text{','} \parallel \ldots \parallel \text{','} \parallel \Gamma_R(H_{l'}^{help}) & \text{rekursive Hilfsregeln sind} \\ & \\ \text{'WITH RECURSIVE'} \parallel & \text{falls } R_1^{help}, ..., R_l^{help} \\ \Gamma_R(H_1^{help}) \parallel \text{','} \parallel \ldots \parallel \text{','} \parallel \Gamma_R(H_{l'}^{help}) & \text{rekursive Hilfsregeln sind.} \end{cases}$$

Werden bei Wiederholungsfolgen leere Zeichenketten verknüpft, so werden verbindende Operatoren nicht konkateniert, so wird z.B. $'abc' \parallel \text{','} \parallel \ldots \parallel \text{','} \parallel '\,'$ zu $'abc'$ ohne weitere Kommazeichen konkateniert. Die logische Konstante TRUE dient in der WHERE-Klausel als Platzhalter, damit bei den nachfolgenden Formeln nicht zwischen der ersten und allen weiteren Formeln unterschieden werden muß.

Die Abbildung (Funktion Γ_{Wneg}) eines in einer Formel B negativ auftretenden DB-Literals auf genau eine geschachtelte NOT EXISTS-Formel mit nur einem Relationssymbol in der FROM-Klausel ist nur aufgrund der Bereichsbeschränkung von Datalog-Formeln unproblematisch. Bei der gewählten Darstellungsform, werden Datalog-Variablen, die in mehreren negativen Literalen auftreten, nicht in NSQL-Identitätsvergleiche transformiert. Aufgrund der Bereichsbeschränkung von Datalog-Regeln liegt für solche Variablen immer eine positive Bindung vor, so daß durch diese vereinfachte Abbildung die Ausdruckskraft nicht eingeschränkt und die Semantik nicht modifiziert wird.

Definition 3.2.3 (Γ-Regel)
Sei $\{R_1, ..., R_l\} \subseteq \mathcal{R}$ eine Menge deduktiver Datalog-Regeln der Form

$$R_i := H_i(t_1, \ldots, t_n) \leftarrow B \text{ WITH } R_1^{help}, ..., R_{l'}^{help}, \quad (1 \leq i \leq l, l' \geq 0)$$

mit dem gleichen Kopfliteral $h = rel(\{\bigcup_{i=1}^{l} H_i\})$. Die Menge aller zu den Regeln $R_1, ...R_l$ gehörenden Hilfsregeln sei $\{R_1^{help}, ..., R_{l''}^{help}\} \subseteq \mathcal{RH}$ und $\bigcup_{j=1}^{l''} rel(R_j^{help})$ die Menge ihrer Relationssymbole ($l'' \geq 0$).

- *$\Gamma_{CreateR}$ erzeugt auf der Basis eines Schema $\mathcal{S}^{\mathcal{D}}$ für alle Regeln eines gegebenen Relationssymbols H^{sym} einen NSQL-DDL-Befehl zum Erzeugen einer Sicht.*

$$\Gamma_{CreateR}(H^{sym}) := \begin{cases} \text{'CREATE RECURSIVE VIEW'} & \text{falls} \\ \parallel \Gamma_R(H^{sym}) & R_1, ..., R_l, R_1^{help}, ..., R_{l''}^{help} \\ & \text{rekursiv sind} \\ & \\ \text{'CREATE VIEW'} \parallel \Gamma_R(H^{sym}) & \text{sonst.} \end{cases}$$

- *Γ_R bildet auf der Basis von $\mathcal{S}^{\mathcal{D}}$ die Regeln $R_1, ...R_l$ ($l \geq 1$) für ein gegebenes Relationssymbol H^{sym} auf eine NSQL-Sichtspezifikation ab. Die NSQL-WITH-Klausel ist die Zusammenfassung der Hilfsregeln aller WITH-Klauseln von $R_1, ...R_l$.*

$$\Gamma_R(H^{sym}) := \Gamma_{RHead}(H^{sym}) \parallel \Gamma_{WITH}([rel(R_1^{help}), ..., rel(R_{l''}^{help})]) \parallel \Gamma_U(H^{sym}).$$

- Γ_{RHead} *bildet unter Zugriff auf das DB-Schema* $\mathcal{S}^{\mathcal{D}}$ *ein gegebenes Relationssymbol* H^{sym} *auf die Spezifikation einer NSQL-Sicht ab. Die Anzahl der Attribute von* H^{sym} *sei* n $(n \geq 0)$.

$$\Gamma_{RHead}(H^{sym}) \;:=\; \begin{cases} H^{sym} \| \text{ '(' } \| \\ \Gamma_{attr}(H^{sym},1) \| \ldots \| \Gamma_{attr}(H^{sym},n) \| \text{ ') } AS' & \textit{falls } n > 0 \\ \\ H^{sym} \| \text{ '(' } \| \Gamma_{attr}(H^{sym},1) \| \text{ ') } AS' & \textit{falls } n = 0. \end{cases}$$

- Γ_U *bildet unter Zugriff auf die in* $\mathcal{S}^{\mathcal{D}}$ *vorliegenden Regelspezifikationen die Regelrümpfe* $B_1, \ldots B_l$ *aller Regeln* $R_1, \ldots R_l \in \mathcal{R}$ $(l \geq 1)$ *eines gegebenen Relationssymbols* H^{sym} *auf einen NSQL-UNION-Ausdruck ab.* $t_{j1}, \ldots, t_{jn}$ *seien die Terme des Kopfliterals der Regel* R_j $(1 \leq j \leq l)$.

$$\Gamma_U(H^{sym}) := \Gamma_{SFW}(B_1, [t_{11}, \ldots, t_{1n}]) \,\|\, \text{'UNION'} \| \ldots \| \text{'UNION'} \| \, \Gamma_{SFW}(B_l, [t_{l1}, \ldots, t_{ln}]).$$

Ohne Beschränkung der Ausdruckskraft wird für die Transformation vorausgesetzt, daß im Regelrumpf B einer Integritätsbedingung C nur ein DB-Literal auftritt.

Definition 3.2.4 (Γ-Integritätsbedingung)
Sei $C \in \mathcal{C}$ *eine Integritätsbedingung* $(C := IC \leftarrow B \text{ WITH } R_1^{help}, \ldots, R_{l'}^{help})$ *und* C^{symbol} *das Relationssymbol des Kopfliterals* IC.
$\Gamma_{CreateC}$ *erzeugt für eine gegebene Datalog-Integritätsbedingung* C *einen NSQL-DDL-Befehl zur Erzeugung einer* ASSERTION.

$$\Gamma_{CreateC}(C^{symbol}) \;:=$$

$$\begin{cases} \text{'CREATE ASSERTION' } \| C^{symbol} \| \text{ 'CHECK (' } \| \\ \text{'NOT EXISTS' } \| \text{ '(' } \| \, \Gamma_{WITH}([rel(R_1^{help}), \ldots, rel(R_{l'}^{help})]) \, \| \\ \Gamma_U(C^{symbol}) \| \text{ '))'} & \textit{falls } B = \neg L \\ \\ \text{'CREATE ASSERTION' } \| C^{symbol} \| \text{ 'CHECK (' } \| \\ \text{'EXISTS' } \| \text{ '(' } \| \, \Gamma_{WITH}([rel(R_1^{help}), \ldots, rel(R_{l'}^{help})]) \, \| \\ \Gamma_U(C^{symbol}) \| \text{ '))'} & \textit{falls } B = L. \end{cases}$$

Definition 3.2.5 (Γ-Anfrage)
Sei $Q := \{ \; (t_1, \ldots, t_n) \mid B \text{ WITH } R_1^{help}, \ldots, R_{l'}^{help} \; \}$ *eine Datalog-Anfrage* $(n \geq 0, \; l' \geq 0)$ *bzgl. eines Schemas* $\mathcal{S}^{\mathcal{D}}$.
Γ_Q *bildet eine gegebene Datalog-Anfrage* Q *auf eine NSQL-Anfrage ab.*

$$\Gamma_Q(Q) \;:=\; \Gamma_{WITH}([rel(R_1^{help}), \ldots, rel(R_{l'}^{help})]) \,\|\, \Gamma_{SFW}(B, [t_1, \ldots, t_n]).$$

Beispiel 3.2.6 (Sicht: mita_mgr)
Seien die beiden Regeln O_1, O_2 für die Datalog-Sicht mita_mgr gegeben. Die Variable O enthalte das Relationssymbol des Kopfliterals der Regeln ($O :=$ 'mita_mgr').

$$\begin{aligned} O_1 &:= mita\_mgr(X_1, X_3, X_4) \leftarrow mita\_abt(X_1, X_2, X_3, X_4); \\ O_2 &:= mita\_mgr(X_1, X_2, Y_4) \leftarrow mita\_mgr(X_1, X_2, X_3) \\ &\qquad\qquad \wedge\, mita\_abt(X_3, Y_2, Y_3, Y_4); \end{aligned}$$

Durch Anwendung der Funktion $\Gamma_{CreateR}(O)$ wird die Regelspezifikation in einen Befehl zur Erzeugung einer SQL-Sicht transformiert.

$$\begin{aligned} \Gamma_{CreateR}(O) \;:=\; & \text{'}\mathit{CREATE\ RECURSIVE\ VIEW}\ mita\_mgr(a_1, a_2, a_3)\ \mathit{AS} \\ & \quad \mathit{SELECT}\ tv_{ma}.a_1, tv_{ma}.a_3, tv_{ma}.a_4 \quad \mathit{FROM}\ \ mita\_abt\ \mathit{AS}\ tv_{ma} \\ & \quad \mathit{UNION} \\ & \quad \mathit{SELECT}\ tv_{mm}.a_1, tv_{mm}.a_2, tv_{ma}.a_4 \\ & \quad \mathit{FROM}\ \ mita\_mgr\ \mathit{AS}\ tv_{mm},\ mita\_abt\ \mathit{AS}\ tv_{ma} \\ & \quad \mathit{WHERE\ TRUE\ AND}\ tv_{mm}.a_3 = tv_{ma}.a_1\text{'} \end{aligned}$$

wobei:

$$\begin{aligned} &\Gamma_{CreateR}(O) \;:=\; \text{'}\mathit{CREATE\ RECURSIVE\ VIEW}\text{'} \,\|\Gamma_R(O) \\ &\Gamma_R(O) \;:=\; \text{'}mita\_mgr(a_1, a_2, a_3)\ \mathit{AS}\text{'} \,\| \\ &\qquad \Gamma_{SFW}(O_1, [X_1, X_2, X_3]) \,\|\, \text{'}\mathit{UNION}\text{'} \,\|\, \Gamma_{SFW}(O_2, [X_1, X_2, Y_4]) \\ &\Gamma_{SFW}(O_1, [X_1, X_2, X_3]) \;:=\; \text{'}\mathit{SELECT}\ tv_{ma}.a_1, tv_{ma}.a_3, tv_{ma}.a_4 \\ &\qquad\qquad \mathit{FROM}\quad mita\_abt\ \mathit{AS}\ tv_{ma}\text{'} \\ &\Gamma_{SFW}(O_2, [X_1, X_2, Y_4]) \;:=\; \text{'}\mathit{SELECT}\ tv_{mm}.a_1, tv_{mm}.a_2, tv_{ma}.a_4 \\ &\qquad\qquad \mathit{FROM}\quad mita\_mgr\ \mathit{AS}\ tv_{mm},\ mita\_abt\ \mathit{AS}\ tv_{ma} \\ &\qquad\qquad \mathit{WHERE\ TRUE\ AND}\ tv_{mm}.a_3 = tv_{ma}.a_1\text{'} \end{aligned}$$

Beispiel 3.2.7 (Sicht: lower_mgr)
Sei die Spezifikationen O für die Datalog-Sicht lower_mgr gegeben. Die Variable O enthalte das Relationssymbol des Kopfliterals der Regeln ($O :=$ 'lower_mgr').

$$\begin{aligned} &lower\_mgr(X_1, X_2, X_3) \leftarrow B; \\ &B \;:=\; mita\_mgr(X_1, X_2, X_3) \wedge \neg h_1(X_3)\ \mathit{WITH}\ O^*; \\ &O^* :=\; h_1(Y_1) \;\leftarrow\; mitarbeiter(Y_1, 13, Y_3, Y_4); \end{aligned}$$

Durch Anwendung der Funktion $\Gamma_{CreateR}(O)$ wird die Regelspezifikation in einen Befehl zur Erzeugung einer SQL-Sicht transformiert.

$$\begin{aligned} \Gamma_{CreateR}(O) \;:=\; & \text{'}\mathit{CREATE\ VIEW}\ lower\_mgr(a_1, a_2, a_3)\ \mathit{AS\ WITH} \\ & \quad h_1(a_1)\ \mathit{AS}\ \ \mathit{SELECT}\ tv_m.a_1 \quad \mathit{FROM}\ mitarbeiter\ \mathit{AS}\ tv_m \\ & \qquad\qquad \mathit{WHERE\ TRUE\ AND}\ tv_m.a_2 = 13 \\ & \quad \mathit{SELECT}\ tv_{mm}.a_1, tv_{mm}.a_2, tv_{mm}.a_3 \quad \mathit{FROM}\ mita\_mgr\ \mathit{AS}\ tv_{mm} \\ & \quad \mathit{WHERE\ TRUE\ AND\ \ NOT\ EXISTS}\ (\\ & \qquad\qquad \mathit{SELECT\ 1\ FROM}\ h_1\ \mathit{AS}\ tv_{h_1} \\ & \qquad\qquad \mathit{WHERE\ TRUE\ AND}\ tv_{h_1}.a_1 = tv_{mm}.a_3\ ) \end{aligned}$$

wobei:

$$\begin{aligned} &\Gamma_{CreateR}(O) \;:=\; \text{'}\mathit{CREATE\ VIEW}\text{'} \,\|\Gamma_R(O) \\ &\Gamma_R(O) \;:=\; \text{'}lower\_mgr(a_1, a_2, a_3)\ \mathit{AS}\text{'} \;\|\; \Gamma_{WITH}([\text{'}h_1\text{'}]) \,\|\, \Gamma_{SFW}(B, [X_1, X_2, X_3]) \\ &\Gamma_{WITH}([\text{'}h_1\text{'}]) \;:=\; \text{'}\mathit{WITH}\ h_1(a_1)\ \ \mathit{AS}\text{'} \;\|\; \Gamma_{SFW}(\text{'}h_1(Y_1)\text{'},\ [Y_1]) \end{aligned}$$

$\Gamma_{SFW}(\,'h_1(Y_1)\,',[Y_1]) := \,'$*SELECT* $tv_m.a_1$ *FROM mitarbeiter AS* tv_m
WHERE TRUE$'\|\Gamma_{Wspez}(\,'h_1(Y_1)\,','mitarbeiter(Y_1,Y_2,13,Y_4)\,')$
$\Gamma_{Wspez}(\,'h_1(Y_1)\,',\ 'mitarbeiter(Y_1,Y_2,13,Y_4)\,') := \,'$*AND* $tv_m.a_3 = 13\,'$
$\Gamma_{SFW}(B,\ [X_1,X_2,X_3]) := \,'$*SELECT* $tv_{mm}.a_1, tv_{mm}.a_2, tv_{mm}.a_3$
FROM *mita_mgr AS* tv_{mm}
WHERE TRUE$' \,\|\, \Gamma_{Wneg}(B,'h_1(Y_1)\,')$
$\Gamma_{Wneg}(B,'h_1(Y_1)\,') := \,'$*AND NOT EXISTS* (*SELECT 1 FROM* h_1 *AS* tv_{h_1}
WHERE TRUE AND $tv_{h_1}.a_1 = tv_{mm}.a_3$)$'$

Beispiel 3.2.8 (ASSERTION: ic_2)
Sei O die Spezifikation einer Datalog-Integritätsbedingung, die die Anforderung formuliert, daß Manager der gleichen Abteilung wie die Angestellten oder der Abteilung 'Management' ('MG') angehören. Die Variable C enthalte das Relationssymbol der Integritätsbedingung ($C := 'ic_2\,'$).

$ic_2 \leftarrow \neg h_1$ *WITH* $S_1^{help},\ S_2^{help},\ S_3^{help}$;
$S_1^{help} := h_1 \leftarrow B_1;$ $B_1 := mita\_mgr(X_1,X_2,X_3) \wedge \neg h_2(X_3,X_2);$
$S_2^{help} := h_1 \leftarrow B_2;$ $B_2 := mita\_mgr(X_1,X_2,X_3) \wedge \neg h_2(X_3,'MG');$
$S_3^{help} := h_2(Y_1,Y_2) \leftarrow B_3;$ $B_3 := mita\_abt(Y_1,Y_2,Y_3);$

Der Datalog-Ausdruck C wird durch Anwendung der Funktion $\Gamma_{CreateC}(C)$ in einen SQL-Ausdruck transformiert:

$\Gamma_{CreateC}(C) :=$
$'$*CREATE ASSERTION* ic_1 *CHECK* (*NOT EXISTS* (*WITH*
$h_1(a_1)$ *AS SELECT* 1 *FROM* $mita\_mgr$ *AS* tv_{mm}
WHERE TRUE AND NOT EXISTS (*SELECT 1 FROM* h_2 *AS* tv_{h_2}
WHERE TRUE AND $tv_{h_2}.a_1 = tv_{mm}.a_3$
AND $tv_{h_2}.a_2 = tv_{mm}.a_2$)
UNION
SELECT 1 *FROM* $mita\_mgr$ *AS* tv_{mm}
WHERE TRUE AND NOT EXISTS (*SELECT 1 FROM* h_2 *AS* tv_{h_2}
WHERE TRUE AND $tv_{h_2}.a_1 = tv_{mm}.a_3\,'$
AND $tv_{h_2}.a_2 =$ *'MG'*),
$h_2(a_1,a_2)$ *AS SELECT* $tv_{ma}.mita,\ tv_{ma}.bez$
FROM $mita\_abt$ *AS* tv_{ma}
SELECT 1 *FROM* h_1 *AS* $tv_{h_1}\,'$.

wobei:

$\Gamma_{CreateC}(C) := \,'$*CREATE ASSERTION*$' \,\|\, '\,ic_1\,' \,\|\,$ *CHECK* (*NOT EXISTS* ($'$
$\|\,\Gamma_{WITH}([\,'h_1\,','h_2\,'])\,\|\,\Gamma_U(\,'ic_2 \leftarrow h_1\,')\,\|\,'\,)\,)\,'$
$\Gamma_{WITH}([\,'h_1\,','h_2\,']) := \,'$*WITH*$' \,\|\,\Gamma_R(\,'h_1\,')\,\|\,',\|\,\Gamma_R(\,'h_2\,')$
$\Gamma_R(\,'h_1\,') := \,'h_1(a_1)$ *AS*$' \,\|\,\Gamma_{SFW}(B_1,[\_]) \,\|\,$ *'UNION'* $\|\,\Gamma_{SFW}(B_2,[\_])$
$\Gamma_{SFW}(B_1,[\_]) := \,'$*SELECT* 1 *FROM* $mita\_mgr$ *AS* tv_{mm}
WHERE TRUE$' \,\|\,\Gamma_{Wneg}(B_1,\ 'h_2(X_3,X_2)\,')$
$\Gamma_{Wneg}(B_1,\ 'h_2(X_3,X_2)\,') := \,'$*AND NOT EXISTS* (
SELECT 1 FROM h_2 *AS* tv_{h_2}
WHERE TRUE AND $tv_{h_2}.a_1 = tv_{mm}.a_3$
AND $tv_{h_2}.a_2 = tv_{mm}.a_2$)$'$

$$\Gamma_{SFW}(B_2, [\_]) := \text{'SELECT 1 FROM } mita\_mgr \text{ AS } tv_{mm} \text{ WHERE TRUE '} \parallel \Gamma_{Wneg}(B_2, \text{ 'h}_2(X_3, \text{'MG'})\text{'})$$

$$\Gamma_{Wneg}(B_2, \text{ 'h}_2(X_3, \text{'MG'})\text{'}) := \text{'AND NOT EXISTS (SELECT 1 FROM } h_2 \text{ AS } tv_{h_2} \text{ WHERE TRUE AND } tv_{h_2}.a_1 = tv_{mm}.a_3 \text{ AND } tv_{h_2}.a_2 = \text{'MG')'}$$

$$\Gamma_R(\text{'h}_2\text{'}) := \text{'h}_2(a_1, a_2) \text{ AS SELECT } tv_{ma}.mita, \ tv_{ma}.bez \text{ FROM } mita\_abt \text{ AS } tv_{ma}\text{'}$$

$$\Gamma_U(\text{'}ic_2 \leftarrow h_1\text{'}) := \text{'SELECT 1 FROM } h_1 \text{ AS } tv_{h_1}\text{'}.$$

3.2.2 Transformation NSQL → Datalog

Ausgehend von einem NSQL-DB-Schema $\mathcal{S}^{\mathcal{S}}$ werden mit Hilfe von Ψ-Funktionen die Spezifikationen von Sichten, ASSERTIONs und Anfragen in Anweisungen zur Erzeugung sicherer Datalog-Regeln, -Integritätsbedingungen bzw. -Anfragen transformiert. Analog zu den Γ-Funktionen ist die Gesamtaufgabe der Transformation in verschiedene Einzelfunktionen unterteilt, deren Ergebnisse (Zeichenketten) mit einander konkateniert werden (‖). Da sich bei den Ψ- wie auch den Γ-Funktionen die Indizierung im wesentlichen an den bearbeiteten NSQL-Teilausdrücken orientiert, werden die Γ- und Ψ-Funktionen inverser Transformationsschritte mit der gleichen Indizierung gekennzeichnet. Die Aufrufe der verschiedenen Ψ-Funktionen werden in der Abbildung 3.2 skizziert. Um eine lesbare Darstellung zu gewährleisten, sind die Aufrufe für die beiden Funktionen Ψ_{sicht} und Ψ_{term} nur angedeutet, denn diese beiden Funktionen werden von einem Großteil der anderen Funktionen aufgerufen. Wie auch bereits bei den Γ-Funktionen gibt es auch bei den Ψ-Funktionen sogenannte Stern-Funktionen mit analoger Funktionalität, auf deren Darstellung auch hier verzichtet wird.

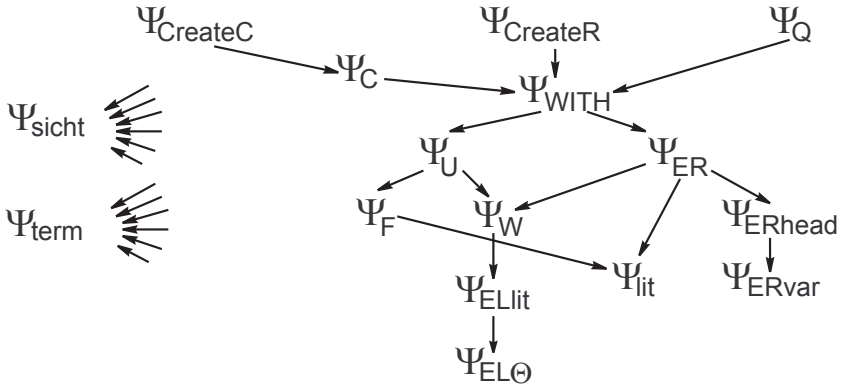

Abbildung 3.2: Transformationsfunktionen NSQL ⟶ Datalog

- Mittels der Funktionen $\mathbf{\Psi_{CreateR}}$, $\mathbf{\Psi_{CreateC}}$ und $\mathbf{\Gamma_Q}$ wird eine Sicht, ASSERTION oder NSQL-Anfrage in einen Datalog-Befehl zur Erzeugung einer bereichsbeschränkten Regel, Integritätsbedingung oder in eine Datalog-Anfrage transformiert. Ψ_C bildet

eine SQL-ASSERTION auf eine Regel ab, die dann als Datalog-Integritätsbedingung definiert wird.

- Die **Ψ-Transformationsfunktionen** Ψ_{WITH}, Ψ_U, Ψ_F, Ψ_W bilden SFW-Ausdrücke in Datalog-Hilfsregeln ab. Ψ_{WITH} generiert eine Datalog-WITH-Klausel für eine NSQL-WITH-Klausel, für Unteranfragen in der WHERE-Klausel und durch UNION-Operator verknüpfte SFW-Ausdrücke. Die UNION verknüpften SFW-Ausdrücke werden mittels der Ψ_U-Funktion in eine Menge von Hilfsregeln transformiert. Die FROM- und WHERE-Klauseln werden durch die Ψ_F- und Ψ_W-Funktionen auf Datalog-Rumpfliterale abgebildet.

- NSQL-Unteranfragen aus einer WHERE-Klausel werden mittels der **Ψ-EXISTS-Funktionen** Ψ_{ER}, Ψ_{ERhead}, Ψ_{ERvar}, Ψ_{ELlit}, $\Psi_{EL\Theta}$ auf Datalog-Hilfsregeln abgebildet. Ψ_{ER} tranformiert alle Unteranfragen in eine Liste von Hilfsregeln. Ψ_{ERhead} generiert das im Kontext der Regel, Integritätsbedingung oder Anfrage eindeutige Relationssymbol der Hilfsregel. Ψ_{ERvar} bildet einen NSQL-Attributterm auf einen Datalog-Term ab. Die Rumpfliterale der Hilfsregeln bildet Ψ_{ELlit} auf der Basis der Teilformeln einer WHERE-Klausel ab, wobei $\Psi_{EL\Theta}$ eine Teilformel auf ein Rumpfliteral abbildet.

- Die **Ψ-Namensfunktionen** Ψ_{rel}, Ψ_{sicht}, Ψ_{term}, Ψ_{lit} bilden NSQL-Attributterme auf Datalog-Terme ab. Ψ_{rel} bildet eine Tupelvariable auf ein Relationssymbol ab. Für eine Anfrage oder Formel generiert Ψ_{sicht} das Relationssymbol einer Hilfsregel. Mittels Ψ_{term} wird ein Attributterm auf einen Datalog-Term abgebildet und mittels Ψ_{lit} ein Relationssymbol auf ein Literal.

Im weiteren sei eine NSQL-DB $\mathcal{D}^{\mathcal{S}}$ mit einem Schema $\mathcal{S}^{\mathcal{S}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C}, \mathcal{T} \rangle$ und der Signatur $\Sigma^S = \langle \mathcal{PS}, \mathcal{AS}, \mathcal{CS}, \mathcal{TS} \rangle$ gegeben. Für alle Funktionen ist das Schema $\mathcal{S}^{\mathcal{S}}$ impliziter Eingabeparameter, der aus Gründen der Übersichtlichkeit nicht aufgeführt wird.

Definition 3.2.9 (Ψ-Namensfunktionen)

- *Ψ_{rel} bildet eine gegebene Tupelvariable T_R auf das in einer FROM-Klausel zugeordnete Relationssymbol ($r \in rel(\mathcal{R})$) ab.*

$$\Psi_{rel}(T_R) \;:=\; r.$$

- *Sei O ein gegebener NSQL-Anfrageausdruck oder eine Formel. Dann generiert Ψ_{sicht} einen im Kontext von O eindeutiges Relationssymbol für eine Hilfssicht ($h \in \mathcal{RH}$).*

$$\Psi_{sicht}(O) \;:=\; h.$$

- *Sei t ein gegebener NSQL-Term. Ist t eine Konstante, dann wird t von Ψ_{term} auf die Konstante abgebildet. Ist t der Term für ein Attribut A_j der Relation R mit der Attributposition j, dann bildet Ψ_{term} t auf eine Datalog-Bereichsvariable X_j^R ab.*

$$\Psi_{term}(t) \;:=\; \begin{cases} X_j^R & \text{falls } t = T_R.A_j \;\wedge\; A_j \in \mathcal{AS} \wedge \Psi_{rel}(T_R) \in \mathcal{PS} \\ t & \text{falls } t \text{ eine Konstante ist.} \end{cases}$$

Sei t ein gegebenes NSQL-Attributsymbol oder eine Konstante. Ist t eine Konstante, dann bildet Ψ_{term} t auf die Konstante ab, ist t ein Attributsymbol dann auf eine Datalog-Bereichsvariable X_j^O. O sei eine gegebene Sicht, Anfrage oder Formel und wird als Kontext bezeichnet, in dem X_j^O eindeutig ist.

$$\Psi_{term}(O,t) \;:=\; \begin{cases} X_j^O & \textit{falls } \; t \in \mathcal{AS} \\ \\ t & \textit{falls } \; t \textit{ eine Konstante ist.} \end{cases}$$

- *Ψ_{lit} bildet ein gegebenes Relationssymbol $R \in rel(\mathcal{B} \cup \mathcal{R})$ auf das zugehörige Datalog-Literal ab. $a_1, ..., a_n$ seien die Attributsymbole von R $(n \geq 1)$.*

$$\Psi_{lit}(R) \;:=\; R \| \text{'('} \| \Psi_{term}(R, a_1) \| \text{','} \| ... \| \text{','} \| \Psi_{term}(R, a_n) \| \text{')'}.$$

Die Namensgleichheit der beiden Ψ_{term}-Funktionen ist ausdrücklich erwünscht, da die Aufgabe beider Funktionen die Abbildung von SQL-Termen bzw. Attributtermen auf Datalog-Bereichsvariablen ist. Die Funktionen werden aufgrund ihrer Parameterstruktur unterschieden. $\Psi_{term}(t)$ bildet die Terme von Attributen, die in NSQL-Ausdrücken auftreten auf Datalog-Bereichsvariablen ab und $\Psi_{term}(O,t)$ die Attributpositionen, für die im NSQL-Ausdruck keine Attributterme referenziert werden, denen jedoch aufgrund der positionellen Darstellung in Datalog Bereichsvariablen zugeordnet werden müssen.

SFW-Ausdrücke werden prinzipiell mit der Ψ_U-Funktion in Hilfsregeln transformiert. Handelt es sich bei dem SFW-Ausdruck um einen UNION-Ausdruck, so sind diese Hilfsregeln erforderlich, um die Mengensemantik in Datalog mittels mehrerer Regeln für ein Relationssymbol ausdrücken zu können. Ist lediglich ein SFW-Ausdruck gegeben, so ist die Bildung einer Hilfssicht nicht erforderlich. Um die Anzahl der zu definierenden Transformationsregeln gering zu halten, wird diese Hilfsregel jedoch im Kauf genommen.

Definition 3.2.10 (Ψ-Transformationsfunktionen)

- *Sei $T := R_1$ AS T_{R_1} , ..., R_k AS T_{R_k} eine gegebene NSQL-FROM-Klausel mit den Zuordnungen von Tupelvariablen T_{R_i} zu Relationssymbolen R_i $(1 \leq i \leq k)$. Ψ_F bildet T auf eine Konjunktion positiver Datalog-DB-Literale ab.*

$$\Psi_F(T) \;:=\; \Psi_{lit}(R_1) \| \text{'}\wedge\text{'} \| \ldots \| \text{'}\wedge\text{'} \| \Psi_{lit}(R_k).$$

- *Sei $F := F_1$ AND...AND F_l eine gegebene NSQL-Formel der WHERE-Klausel eines NSQL-Anfrageausdrucks O. Ψ_W bildet F auf eine Konjunktion von Datalog-DB- und -Vergleichsliteralen ab.*

$$\Psi_W(O,F) \;:=\; \text{'}\wedge\text{'} \| \Psi_{W^*}(O, F_1) \| \text{'}\wedge\text{'} \| ... \| \text{'}\wedge\text{'} \| \Psi_{W^*}(O, F_l).$$

– *Sei F^* eine existenzquantifizierte NSQL-Formel eines Anfrageausdrucks O oder eine atomare NSQL-Vergleichsformel mit den Termen t_j und t_k, die entweder Konstanten oder Attributterme sind, die in der FROM-Klausel von O gebunden sind, dann bildet*
Ψ_{W^*} die Teilformel F^* auf ein Datalog-DB-Literal und ggf. mehrere Vergleichsatome oder auf ein Datalog-Vergleichsliteral ab.

$$\Psi_{W^*}(O, F^*) := \begin{cases} \Psi_{ELlit}(O, O^*) & \text{falls } F^* = \textit{EXISTS } (O^*) \\ \text{'}\neg\text{'} \| \Psi_{ELlit}(O, O^*) & \text{falls } F^* = \textit{NOT EXISTS } (O^*) \\ \Psi_{term}(t_j) \| \Theta \| \Psi_{term}(t_k) & \text{falls } F^* = t_j \Theta t_k \\ \text{'}\neg\text{'} \| \Psi_{term}(t_j) \| \Theta \| \Psi_{term}(t_k) & \text{falls } F^* = \textit{NOT } (t_j \Theta t_k). \end{cases}$$

- *Sei $O := O_1^*$ UNION...UNION O_l^* ein gegebener NSQL-Anfrageausdruck mit SFW-Ausdrücken O_i^* $(1 \leq i \leq l)$ verknüpft durch UNION-Operatoren falls $l > 1$. Ψ_U bildet die SFW-Ausdrücke $O_1^*, ..., O_l^*$ auf eine Liste von Datalog-Hilfsregeln ab.*

$$\Psi_U(O) := \Psi_{U^*}(O, O_1^*) \| \text{','} \| ... \| \text{','} \| \Psi_{U^*}(O, O_l^*).$$

– *Sei O^* ein SFW-Ausdruck, der in einem gegebenen Anfrageausdruck O auftritt. $t_1^*, ..., t_n^*$ seien die Terme der SELECT-Liste von O^*, T^* sei die FROM-Klausel und F^* die NSQL-Formel der WHERE-Klausel von O^*.*
Ψ_{U^*} bildet einen NSQL-Anfrageausdruck O^* eines (UNION-)Ausdrucks O auf eine Datalog-Hilfsregel ab. Das Relationssymbol des Kopfliteral der Hilfsregel ist für alle Teilanfragen von O gleich und wird daher für O generiert (Kontext).

$$\begin{aligned} \Psi_{U^*}(O, O^*) := \Psi_{sicht}(O) \| \text{'('} \| \Psi_{term}(t_1^*) \| \text{','} \| ... \| \text{','} \| \Psi_{term}(t_n^*) \| \text{')'} \\ \| \text{' } \leftarrow \text{ '} \| \Psi_F(T^*) \| \Psi_W(O^*, F^*). \end{aligned}$$

- *Sei O := WITH [RECURSIVE] $R_1^{help}, ..., R_l^{help}$ Q ein gegebener NSQL-WITH-Anfrageausdruck mit den Hilfssichten $R_1^{help}, ..., R_l^{help}$ $(l \geq 0)$ und dem Anfrageausdruck Q.*
Ψ_{WITH} bildet O auf eine Datalog-WITH-Klausel ab. Eine NSQL-Hilfssicht R_i^{help} $(0 \leq i \leq l)$ wird durch Ψ_{WITH^*} in eine Datalog-Hilfsregel transformiert. Existenzquantifizierte Unteranfragen von Q werden durch Ψ_{ER} und der Anfrageausdruck Q durch Ψ_U ebenfalls auf Hilfsregeln abgebildet.

$$\begin{aligned} \Psi_{WITH}(O) := \text{'WITH'} \| \Psi_{WITH^*}(R_1^{help}) \| \text{','} \| ... \| \text{','} \| \Psi_{WITH^*}(R_l^{help}) \\ \| \text{','} \| \Psi_{ER}(Q) \| \text{','} \| \Psi_U(Q). \end{aligned}$$

– *Sei R^{help} := $S(A_1, ..., A_n)$ AS Q eine gegebene Hilfssicht, dann bildet Ψ_{WITH^*} die Hilfssicht R^{help} auf eine Datalog-Hilfsregel ab. Ist die Anfrage Q ein UNION-Ausdruck oder enthält die WHERE-Klausel von Q EXISTS-Formeln, so ist das Ergebnis eine Liste von Hilfsregeln (Ψ_U, Ψ_{ER}).*

$$\begin{aligned}\Psi_{WITH^*}(R^{help}) \;:=\; & S\|\,'('\|\;\Psi_{term}(S,A_1)\|\,',\|...\|\,',\|\Psi_{term}(S,A_n)\|\,')\,'\|\,'\leftarrow' \\ & \|\Psi_{sicht}(Q)\|\,'('\|\;\Psi_{term}(S,A_1)\|\,',\|...\|\,',\|\Psi_{term}(S,A_n)\|\,')' \\ & \|\,',\|\Psi_U(Q)\|\;\;',' \;\|\;\Psi_{ER}(Q).\end{aligned}$$

Eine NSQL-EXISTS-Formel muß auf eine Datalog-Hilfsregel abgebildet werden ('$_{ER}$' indizierte Funktionen) und auf ein positives oder negatives Datalog-DB-Atom ('$_{EL}$' indizierte Funktionen), das dann im Rumpf einer Datalog-Anfrage oder -Regel auftritt. In quantifizierten NSQL-Unteranfragen tritt in der FROM-Klausel ohne Einschränkung der Ausdruckskraft nur ein Relationssymbol und eine Tupelvariable auf.

Definition 3.2.11 (Ψ-EXISTS-Funktionen)
Sei O ein gegebener Anfrageausdruck mit $O_1,...,O_l$ $(l \geq 0)$ existenzquantifizierten NSQL-Unteranfragen in der WHERE-Klausel.
Sei $O^* :=$ [NOT] EXISTS (Q) eine gegebene NSQL-Formel aus der WHERE-Klausel von O $(O^* \in \{O_1,...,O_l\})$.

- *Ψ_{ER} bildet $O_1,...,O_l$ auf eine Liste von Datalog-Hilfsregeln ab (Integration in die WITH-Liste).*

  $$\Psi_{ER}(O) \;:=\; \Psi_{ER^*}(O,O_1)\;\|\,',\|...\|\,',\|\Psi_{ER^*}(O,O_l).$$

  – *L sei das Relationssymbol in der FROM-Klausel von Q und F die Formel der WHERE-Klausel von Q. Ψ_{ER^*} bildet O^* auf eine Datalog-Hilfsregel ab.*

  $$\Psi_{ER^*}(O,O^*) \;:=\; \Psi_{ERhead}(O,O^*)\|\,'\leftarrow'\,\|\Psi_{lit}(L)\|\,'\wedge'\|\Psi_W(O^*,F).$$

- *$t_1,...,t_n$ $(0 \leq n)$ seien die Attributterme der Unteranfrage Q in O^*, die durch Vergleichsatome in der WHERE-Klausel von Q mit Attributtermen der übergeordneten Anfrage O verglichen werden, dann bildet*
  Ψ_{ERhead} O^* auf das Kopfliteral einer (ggf. parameterlosen) Hilfsregel ab.

  $$\Psi_{ERhead}(O,O^*) := \begin{cases} \Psi_{sicht}(O^*)\|\,'('\|\Psi_{ERvar}(O,O^*,t_1)\|\,',\|...\|\,',' \\ \qquad\qquad \|\Psi_{ERvar}(O,O^*,t_n)\|\,')' & \text{falls } n \geq 0 \\ \Psi_{sicht}(O^*) & \text{sonst.} \end{cases}$$

- *$F := F_1$ AND...AND F_m sei die WHERE-Klausel der Unteranfrage Q in O^* und t sei ein gegebener Attributterm der SELECT-Liste von Q.*
  Ψ_{ERvar} bildet t auf eine Datalog-Bereichsvariable ab, wenn t in einer Vergleichsformel F_i $(1 \leq i \leq m)$ mit einem Attributterm verglichen wird, dessen Tupelvariable in der übergeordneten FROM-Klausel von O gebunden ist (positive Bindung).

  $$\Psi_{ERvar}(O,O^*,t) \;:=\; \Psi_{term}(t) \qquad \textit{falls}\;\; \exists\, F_i \in \{F_1,...,F_m\} : \Psi_{EL\Theta}(F_i) \neq\; ' \,'.$$

- *$F := F_1$ AND...AND F_m sei die* WHERE*-Klausel der Unteranfrage Q in O^* und $[t_1, ..., t_n]$ sei die Liste der Attributterme von F, deren Tupelvariablen in der* FROM*-Klausel der übergeordneten Anfrage O gebunden ist.*
  Ψ_{ELlit} bildet eine quantifizierte Formel O^* auf ein (parameterloses) Datalog-DB-Atom und ggf. Vergleichsatome ($\Psi_{EL\Theta}$) ab. Als Parameter des DB-Atoms werden nur die NSQL-Attributterme ($[t_1, ..., t_n]$) von O^* auf Datalog-Bereichsvariablen abgebildet, die in der übergeordneten Anfrage O gebunden sind. Die Vergleichsatome gewährleisten diese positive Bindung von $t_1, ..., t_n$ in der Datalog-Regel.

$$\Psi_{ELlit}(O, O^*) := \begin{cases} \Psi_{sicht}(O^*)\| & \\ \text{'('}\|\Psi_{term}(t_1)\|\text{','}\|...\|\text{','}\|\Psi_{term}(t_n)\|\text{')'} & \\ \|\text{'}\wedge\text{'}\|\Psi_{EL\Theta}(F_1)\|\text{'}\wedge\text{'}\|...\|\text{'}\wedge\text{'}\|\Psi_{EL\Theta}(F_m) & \\ & \\ \Psi_{sicht}(O^*) & \end{cases} \begin{array}{l} \textit{falls } n > 0 \\ \\ \\ \\ \textit{sonst.} \end{array}$$

  - *Sei $t_j \in [t_1, ..., t_n]$ ein Attributterm aus der Liste der Terme, die mit Termen aus O gebunden sind, und t_k sei ein Attributterm, dessen Tupelvariable in O gebunden ist, dann bildet*
    *$\Psi_{EL\Theta}$ eine gegebene Vergleichsformel $F := [$*NOT*$]\ (t_j \Theta t_k)$ von O^* auf ein Datalog-Vergleichsatom ab. Den NSQL-Attributtermen werden die Bereichsvariablen zugeordnet, die auch beim zugehörigen DB-Atom im Regelrumpf verwendet werden. Hinsichtlich dieser Transformation ist Θ kommutativ.*

$$\Psi_{EL\Theta}(F) := \begin{cases} \Psi_{term}(t_j)\|\Theta\|\Psi_{term}(t_k) & \textit{falls } F = t_j \Theta t_k \\ \\ \text{'}\neg\text{'}\|\Psi_{term}(t_j)\|\Theta\|\Psi_{term}(t_k) & \textit{falls } F = \textit{NOT } (t_j \Theta t_k). \end{cases}$$

Bei $\Psi_{EL\Theta}$ ist zu beachten, daß nur die NSQL-Teilformeln einer WHERE-Klausel auf Datalog-Vergleichsliterale abgebildet werden, die Vergleiche zwischen Attributtermen der Unteranfrage und der übergeordneten Anfrage darstellen. Die Transformation der NSQL-Teilformeln, die Vergleiche von Attributtermen der Unteranfrage mit Konstanten oder anderen Attributtermen der Unteranfrage darstellen, werden in Rumpfliterale der Hilfsregel übersetzt (Ψ_W-Aufruf in Ψ_{ER^*}).

Da in einem NSQL-Schema nur virtuelle Sichten vorliegen, werden nur virtuelle Datalog-Sichten generiert. Der Anfrageausdruck Q einer gegebenen Sicht S wird grundsätzlich auf Hilfssichten abgebildet. Ist Q ein UNION-Ausdruck, so ist die Abbildung auf eine Menge von Hilfssichten notwendig. Aus darstellungstechnischen Gründen wird auch ein SFW-Ausdruck ohne Vereinigungsoperator auf eine Hilfssicht abgebildet, prinzipiell könnte aber ein solcher Ausdruck direkt in die Literale eines Regelrumpfs übersetzt werden.

Definition 3.2.12 (Ψ-Sicht)
Sei $R := S(A_1, ..., A_n)$ AS Q eine NSQL-Sicht (keine Hilfssicht) des DB-Schemas $\mathcal{S}^{\mathcal{S}}$ mit dem Relationssymbol $rel(S) \in \mathcal{PS}$ $(1 < i \leq n)$.

$$\begin{array}{rcl} \Psi_{CreateR}(R) & := & \textit{'CREATE VIRTUAL VIEW'}\| \\ & & rel(S)\|\text{'('}\|\ \Psi_{term}(R, A_1)\|\text{','}\|...\|\text{','}\|\Psi_{term}(R, A_n)\ \|\text{')}\ \leftarrow\ \text{'} \\ & & \quad \|\ \Psi_{sicht}(Q)\|\text{'('}\|\ \Psi_{term}(R, A_1)\|\text{','}\|...\|\text{','}\|\Psi_{term}(R, A_n)\ \|\text{')'} \\ & & \quad \|\ \Psi_{WITH}(Q). \end{array}$$

Die Definition der Transformationsvorschrift für Integritätsbedingungen $\Psi_{CreateC}$ basiert auf der Annahme aus Kapitel 2.1, daß der Regelrumpf einer Integritätsbedingung nur aus einem parameterlosen Literal besteht. Komplexere Zusammenhänge werden mittels (Hilfs-)Sichten formuliert.

Definition 3.2.13 (Ψ-Integritätsbedingung)
Sei eine NSQL-ASSERTION $C := IC$ *CHECK* ([*NOT*] *EXISTS* (Q)) *des DB-Schemas* $\mathcal{S}^{\mathcal{S}}$ *gegeben, mit* $rel(IC) \in \mathcal{CS}$. $t_1, ..., t_n$ *seien die Terme der SELECT-Klausel von* Q

$$\Psi_{CreateC}(C)) \;:=\; \text{'CREATE CONSTRAINT'} \| \; \Psi_C(C)$$

$$\Psi_C(C) \;:=\; \begin{cases} rel(IC) \| \text{'}\leftarrow\text{'} \| \Psi_{sicht}(Q) \| \Psi_{WITH}(Q) & \text{falls } O = \textit{EXISTS } (Q) \\ rel(IC) \| \text{'}\leftarrow \neg\text{'} \| \Psi_{sicht}(Q) \| \Psi_{WITH}(Q) & \text{falls } O = \textit{NOT EXISTS } (Q). \end{cases}$$

Definition 3.2.14 (Ψ-Anfrage)
Sei $Q :=$ *WITH* $R_1^{help}, ..., R_{l'}^{help}$ Q^* *eine NSQL-Anfrage auf* $\mathcal{D}^{\mathcal{S}}$ *mit optionaler WITH-Klausel* ($l' \geq 0$) *und einem Anfrageausdruck* Q^*. $t_1, ..., t_n$ *seien die Terme der SELECT-Klausel von* Q ($n \geq 0$).

$$\begin{aligned} \Psi_Q(Q) \;:=\; & \text{' \{ (' } \| \; \Psi_{term}(Q, t_1) \| \text{', '} \| ... \| \text{', '} \| \Psi_{term}(Q, t_n) \; \| \text{ ') } \mid \text{ '} \\ & \| \; \Psi_{sicht}(Q) \| \text{' ('} \| \; \Psi_{term}(Q, t_1) \| \text{', '} \| ... \| \text{', '} \| \Psi_{term}(Q, t_n) \; \| \text{ ') \} '} \\ & \| \; \Psi_{WITH}(Q). \end{aligned}$$

Beispiel 3.2.15 (Sicht: mita_mgr)
Sei O *die Spezifikation der Sicht 'mita_mgr' gegeben.*

$$\begin{aligned} O \;&:=\; mita\_mgr \;\; (mita, bez, mgr) \;\; AS \;\; O' \\ O' \;&:=\; WITH \;\; RECURSIVE \;\; O_1^* \;\; Q \\ O_1^* \;&:=\; h_1(mita, bez, mgr) \;\; AS \;\; Q_1^* \\ Q_1^* \;&:=\; SELECT \;\; tv_{h_1}.mita, tv_{h_1}.bez, tv_{ma}.mgr \\ &\quad\; FROM \;\; h_1 \;\; AS \;\; tv_{h_1}, \;\; mita\_abt \;\; AS \;\; tv_{ma} \\ &\quad\; WHERE \;\; tv_{h_1}.mgr = tv_{ma}.mita \\ &\quad\; UNION \; SELECT \;\; tv_{ma}.mita, tv_{ma}.bez, tv_{ma}.mgr \\ &\qquad\quad FROM \;\; mita\_abt \;\; AS \;\; tv_{ma} \\ Q \;&:=\; SELECT \;\; tv_{h_1}.mita, tv_{h_1}.bez, tv_{h_1}.mgr \;\; FROM \;\; h_1 \;\; AS \;\; tv_{h_1}; \end{aligned}$$

Die NSQL-Sichtspezifikation O *wird in einen CREATE VIEW-Befehl transformiert.*

$$\begin{aligned} & \text{'CREATE VIRTUAL VIEW } mita\_mgr(X_1, X_2, X_3) \;\leftarrow\; h_3(X_1, X_2, X_3) \\ & WITH \;\; h_1(X_1, X_2, X_3) \;\leftarrow\; h_2(X_1, X_2, X_3), \\ & \qquad h_2(X_1, X_2, X_3) \;\leftarrow\; mita\_abt(X_1, X_2, X_3, X_4), \\ & \qquad h_2(X_1, X_2, X_3) \;\leftarrow\; h_1(X_1, X_2, X_3) \;\wedge\; mita\_abt(X_3, Y_2, Y_3, Y_4), \\ & \qquad h_3(X_1, X_2, X_3) \;\leftarrow\; h_1(X_1, X_2, X_3) \text{ '} \end{aligned}$$

wobei:

$$\begin{aligned} \Psi_{CreateR}(O) \;:=\; & \text{'CREATE VIRTUAL VIEW'} \\ & \| \; \text{'mita\_mgr'} \| \text{'('} \| \; \text{'}X_1\text{'} \| \text{', '} \| \; \text{'}X_2\text{'} \| \text{', '} \| \; \text{'}X_3\text{'} \| \text{') } \leftarrow \text{ '} \end{aligned}$$

$$
\begin{array}{l}
\quad\quad \| \text{ 'h}_3\text{ ' } \| \text{ '(' } \| \text{ 'X}_1\text{ ' } \| \text{ ',' } \| \text{ 'X}_2\text{ ' } \| \text{ ',' } \| \text{ 'X}_3\text{ ' } \| \text{ ')'} \\
\quad\quad \| \text{ 'WITH' } \| \Psi_{WITH^*}(O', O_1^*) \| \text{ ',' } \| \Psi_U(Q) \\
\Psi_{WITH^*}(O', O_1^*) := \text{ 'h}_1\ (X_1, X_2, X_3) \leftarrow h_2\ (X_1, X_2, X_3),\ \text{' } \| \Psi_U(Q_1^*) \\
\Psi_U(Q_1^*) := \Psi_{U^*}(Q_1^*, \text{'SELECT } tv_{ma}.mita, ...\text{'}) \| \text{',' } \| \\
\quad\quad \Psi_{U^*}(Q_1^*, \text{'SELECT } tv_{h_1}.mita, ...\text{'}) \\
\Psi_U(Q_1^*) := \text{ 'h}_2(X_1, X_2, X_3) \leftarrow mita\_abt(X_1, X_2, X_3, X_4), \\
\quad\quad h_2(X_1, X_2, X_3) \leftarrow h_1(X_1, X_2, X_3) \wedge mita\_abt(X_3, Y_2, Y_3, Y_4)\text{'} \\
\Psi_U(Q) := \text{ 'h}_3(X_1, X_2, X_3) \leftarrow h_1(X_1, X_2, X_3)\text{'}
\end{array}
$$

Beispiel 3.2.16 (Sicht: lower_mgr)
Sei O die Spezifikation der Sicht 'lower_mgr' gegeben.

$$
\begin{array}{l}
O := lower\_mgr\ \ (mita, bez, mgr)\ \textit{AS}\ \ O' \\
O' := \textit{SELECT}\ \ tv_{mm}.mita, tv_{mm}.bez, tv_{mm}.mgr \\
\quad\quad \textit{FROM}\ \ \ mita\_mgr\ \textit{AS}\ \ tv_{mm}\ \ \textit{WHERE}\ \ \textit{NOT EXISTS}\ (\ Q\ ) \\
Q := \textit{SELECT}\ 1\ \ \textit{FROM}\ \ mitarbeiter\ \textit{AS}\ \ tv_m \\
\quad\quad \textit{WHERE}\ tv_m.nr = 13\ \ \textit{AND}\ \ tv_m.mita = tv_{mm}.mgr;
\end{array}
$$

Der SQL-Ausdruck O wird durch Anwendung der Funktion $\Psi_R(O)$ in einen CREATE VIEW-Befehl transformiert:

$$
\begin{array}{l}
\text{'}\textit{CREATE VIRTUAL VIEW}\ lower\_mgr(X_1, X_2, X_3) \leftarrow h_1(X_1, X_2, X_3) \\
\textit{WITH}\ \ h_2(Y_1) \leftarrow mitarbeiter(Y_1, Y_2, Y_3, Y_4) \wedge Y_2 = 13 \\
\quad\quad h_1(X_1, X_2, X_3) \leftarrow mita\_mgr(X_1, X_2, X_3) \\
\quad\quad\quad\quad\quad\quad \wedge \neg h_2(Z_1) \wedge Z_1 = X_3\text{'}
\end{array}
$$

wobei:

$$
\begin{array}{l}
\Psi_{CreateR}(O) := \text{ 'CREATE VIRTUAL VIEW'} \\
\quad\quad \| \text{'lower\_mgr'} \| \text{'(' } \| \text{ 'X}_1\text{ ' } \| \text{ ',' } \| \text{ 'X}_2\text{ ' } \| \text{ ',' } \| \text{ 'X}_3\text{ ' } \| \text{ ')} \leftarrow \text{ '} \\
\quad\quad \| \text{ 'h}_1\text{ ' } \| \text{ '(' } \| \text{ 'X}_1\text{ ' } \| \text{ ',' } \| \text{ 'X}_2\text{ ' } \| \text{ ',' } \| \text{ 'X}_3\text{ ' } \| \text{ ')'} \\
\quad\quad \| \text{ 'WITH' } \| \Psi_{ER}(O') \| \text{ ',' } \| \Psi_U(O') \\
\Psi_{ER}(O') := \text{ 'h}_2(Y_1) \leftarrow mitarbeiter(Y_1, Y_2, Y_3, Y_4) \wedge Y_2 = 13\text{'} \\
\Psi_U(O') := \text{ 'h}_1(X_1, X_2, X_3) \leftarrow mita\_mgr(X_1, X_2, X_3)\text{' } \| \text{ '} \wedge \neg \text{' } \| \Psi_{ELlit}(O', Q) \\
\Psi_{ELlit}(O', Q) := \text{ 'h}_2(Z_1) \wedge Z_1 = X_3\text{'}
\end{array}
$$

Beispiel 3.2.17 (ASSERTION: ic_1)
Sei die NSQL-ASSERTION-Spezifikation O gegeben, die die Inklusionsbeziehung der Werte des Attributs mitarbeiter.abteil zu abteilung.nr beschreibt:

$$
\begin{array}{l}
O := ic_1\ \textit{CHECK}\ (\ \textit{NOT EXISTS}\ (O^*)\ ) \\
O^* := \textit{SELECT 1 FROM}\ \ mitarbeiter\ \textit{AS}\ \ tv_m\ \ \textit{WHERE}\ \ \textit{NOT EXISTS} \\
\quad\quad\quad\quad (\ \textit{SELECT 1 FROM}\ abteilung\ \textit{AS}\ \ tv_a \\
\quad\quad\quad\quad\quad \textit{WHERE}\ tv_a.nr = tv_m.abteil\ );
\end{array}
$$

Der SQL-Ausdruck O wird durch Anwendung der Funktion $\Psi_C(O)$ in einen CREATE CONSTRAINT-Befehl transformiert.

$$\begin{array}{l}
\textit{'CREATE CONSTRAINT } ic_1 \ \leftarrow \ \neg \ h_1 \\
\textit{WITH } \ h_2(Y_1) \ \leftarrow \ abteilung(Y_1, Y_2, Y_3) \\
\qquad\quad h_1 \ \leftarrow \ mitarbeiter(X_1, X_2, X_3, X_4) \ \wedge \ \neg h_2(Z_1) \ \wedge \ Z_1 = X_2,
\end{array}$$

wobei:

$$\begin{array}{lcl}
\Psi_{CreateC}(O) & := & \textit{'CREATE CONSTRAINT'} \parallel \\
 & & \textit{'}ic_1\textit{'} \parallel \textit{'} \leftarrow \ \neg \textit{'} \parallel \textit{'}h_1\textit{'} \parallel \textit{' WITH '} \parallel \Psi_{ER}(O^*) \parallel \textit{','} \parallel \Psi_U(O^*) \\
\Psi_{ER}(O^*) & := & \textit{'}h_2(Y_1)\textit{'} \parallel \textit{'} \leftarrow \textit{'} \parallel \textit{'}abteilung(Y_1, Y_2, Y_3) \\
\Psi_U(O^*) & := & \textit{'}h_1 \ \leftarrow \textit{'} \parallel \Psi_F(\textit{'FROM } mitarbeiter \textit{ AS } tv_m\textit{'}) \parallel \\
 & & \qquad \Psi_W(\textit{'NOT EXISTS (SELECT ... FROM } abteilung \textit{ ...)'}) \\
\Psi_U(O^*) & := & \textit{'}h_1 \ \leftarrow \ mitarbeiter(X_1, X_2, X_3, X_4)\textit{'} \parallel \textit{'} \wedge \neg \ h_2(Z_1) \ \wedge \ Z_1 = X_2\textit{'}.
\end{array}$$

Kapitel 4

Änderungspropagierung in Datalog und Relationaler Algebra

In diesem Kapitel wird ein Überblick über die im Zusammenhang mit Änderungspropagierung seit Anfang der 80er Jahre entwickelten Ideen und Verfahren gegeben. Ziel ist es, die Prinzipien dieser Verfahren zu verdeutlichen und unterschiedliche Lösungen für verschiedene Aufgaben kontrovers zu diskutieren. Vor diesem Hintergrund ist die Struktur des Kapitels so gewählt worden, daß die zu lösenden Aufgaben im Mittelpunkt der Betrachtung stehen und zu jeder Problemstellung unterschiedliche, in der Literatur zu findende Lösungen erläutert werden. Zum leichteren Verständnis wird die Darstellung soweit wie möglich an den jeweiligen in Kapitel 2 vorgestellten sprachlichen Kontext angepaßt. Mit der syntaktischen Anpassung ist keine Modifikation der Semantik verbunden. Intention dieser problembezogenen Darstellung ist die sich in Kapitel 5 anschließende Untersuchung der sich im SQL-Kontext ergebenden Problemstellungen bei der Diskussion der Übertragbarkeit der für den Datalog-Kontext entwickelten Lösungen.

In diesem Kapitel werden inkrementelle Verfahren für verschiedene Aufgabenstellungen detailliert erörtert:

- Integritätsprüfung: [Dec86], [LST86], [DW89], [KSS87], [QS87], [CW90], [Kue91], [Oli91], [BMM92], [LL96],
- Sichtenaktualisierung: [CW91], [BMM92], [GMS93], [CW94], [GL95], [CGLMT96], [Grie97], [BDDF+98], [DS00], [SBLC00]
- Änderungspropagierung mit beiden Verwendungszwecken: [UO92], [TO95], [MT00], [CKLMR97], [RSS96]
- Änderungspropagierung ohne Verwendungszweck: [GrMa94], [Man94], [QW91].

In deduktiven Datenbanken werden von Integritätsprüfungs- und Sichtenaktualisierungsverfahren die Ergebnisse der Änderungspropagierung, die sogenannten induzierten Änderungen, genutzt. Da der Propagierungsprozeß unabhängig von der Verwendung der induzierten Änderungen ist, kann für Integritätsprüfung und Sichtenaktualisierung ein gemeinsamer Propagierungsprozeß ausgeführt werden. Neben den beiden hier diskutierten Verwendungen sind u.a. drei weitere sehr naheliegend. Je komplexer DB-Schemata

sind, um so weniger ist es für einen Anwender überschaubar, welche Konsequenzen eine Basisfaktenänderung mit sich bringt. Zur Beurteilung der Korrektheit einer Basisfaktenänderung ist die Kenntnis aller induzierten Änderungen sehr wesentlich. Zudem können leistungsfähige Triggerkomponenten Trigger nicht nur für Basisfaktenänderungen sondern auch für Änderungen abgeleiteter Fakten aktiviert werden. Auf eine Konsistenzverletzung muß auch nicht in jedem Fall mit dem Zurückrollen der Transaktion reagiert werden. In Abhängigkeit von der fehlerverursachenden (induzierten) Änderung können Maßnahmen zur Fehlerkorrektur durchgeführt werden (Abschnitt 4.3.5).

Inkrementelle Verfahren sind im wesentlichen im Datalog-Kontext und im Kontext relationaler Algebra (RA) entwickelt worden.

- Datalog: [Dec86], [LST86], [KSS87], [BDM88], [Kue91], [Oli91], [BMM92], [UO92], [GMS93], [CW94], [Man94], [UO94], [TO95], [LL96], [MT99], [MT00]
- RA: [QW91], [Man94], [GL95], [CGLMT96], [CKLMR97], [BDDF+98], [DS00], [SBLC00]
- SQL: [CW90], [CW91].

Die übersichtliche Darstellung mittels RA-Ausdrücken eignet sich besonders zur Untersuchung grundlegender Abläufe und Zusammenhänge wie bei [QW91], [Man94]. Griffin/Libkin stellen in [GL95] ein Verfahren zur Propagierung von Duplikaten auf der Basis einer Multimengenalgebra $\mathcal{BA}$ vor. Nachteilig ist lediglich, daß rekursive Ausdrücke nicht mittels RA formulierbar sind. Die Arbeiten [CW91] und [CW90] von Ceri/Widom zeigen hingegen, wie komplex und wenig intuitiv die Formulierung inkrementeller Verfahren im SQL-Kontext sein kann. Aufgrund der Normalisierung von NSQL wird versucht dieses Problem in Kapitel 5 zu vermeiden. Vor diesem Hintergrund haben Bello et.al. in [BDDF+98] und Salem et.al. in [SBLC00] ihre Verfahren im RA-Kontext publiziert, obwohl beide in SQL-DB-Systemen implementiert sind. In den letzten Jahren sind inkrementelle Ansätze insbesondere vor dem Hintergrund einer Implementierung in kommerziellen Systemen ([BDDF+98], [SBLC00]) oder vor dem Hintergrund einer speziellen Anwendung oder Problemstellung wie Data Warehouses oder verteilten Datenbeständen ([LMSS95]) und dem Vorhandensein nur partieller Informationen ([GSUW94]) diskutiert worden.

Übersichtsartikel zu diesen Themen sind nur sehr vereinzelt zu finden. Auch diese Dissertation kann nur einen Einblick geben. Gupta/Mumick haben in [GuMu95] einen Problemraum für die Änderungspropagierung mit dem Ziel der Sichtenaktualisierung beschrieben und ein Klassifikationsschema erstellt. Die Struktur der nachfolgenden Abschnitte orientiert sich an den von ihnen aufgestellten Kriterien. Ihr Schema wird jedoch um einige Punkte erweitert, wie z.B. um die Granularität induzierter Änderungen und um die Auswertungsstrategie. Bry et.al. stellen in [BMM92] nicht nur ein inkrementelles Prüfungsverfahren für statische und dynamische Integritätsbedingungen vor, sie geben auch einen vergleichenden Überblick über andere Integritätsprüfungsverfahren. Im Anschluß an die Definition ihres Propagierungsverfahrens beschreiben Teniente/Olivé in [TO95] weitere Verfahren ausführlich und bewerten sie im Vergleich zu ihrer Methode.

In Abschnitt 4.1 werden anhand von Beispielen Basisstrategien inkrementeller Verfahren eingeführt. Aus diesem allgemeinen Verständnis heraus wird ein generisches Verfahren

definiert, welches grundlegenden Anforderungen an ein inkrementelles Sichtenaktualisierungs- und Integritätsprüfungsverfahren im Datalog-Kontext genügt. In Abschnitt 4.2 werden zentrale Eigenschaften inkrementeller Verfahren herausgegriffen sowie alternative Ansätze diskutiert und bewertet. Anforderungen an ein inkrementelles Verfahren, die nicht im Rahmen des generischen Verfahrens berücksichtigt werden, werden in Abschnitt 4.3 ausführlich erörtert, ebenso wie orthogonale Optimierungskonzepte, die mit dem Ziel weiterer Performanzsteigerungen integriert werden können.

4.1 Basiskonzept

4.1.1 Grundlegende Strategien

Entsprechend der historischen Entwicklung werden die Ideen und Strategien inkrementeller Ansätze anhand der Integritätsprüfung erörtert und anschließend auf den Prozeß der Änderungspropagierung und Sichtenaktualisierung übertragen.

Spezialisierung von Integritätsbedingungen:

Grundlegende Ideen inkrementeller Integritätsprüfungsverfahren haben Nicolas und Bernstein/Blaustein bereits Ende der 70er und Anfang der 80er Jahre im Kontext der Prädikatenlogik formuliert. Zu ihren einflußreichen Arbeiten zählen [Nic82] und [BBC80], [BB82]. Intention beider Ansätze ist, durch Transformation ('simplification', Spezialisierung) der Syntax statischer Integritätsbedingungen die Anwendung der Bedingungen auf die geänderten Fakten zu beschränken und so den Aufwand für die Integritätsprüfung zu minimieren. Diese Spezialisierung setzt eine detaillierte Analyse der Syntax der Integritätsbedingungen voraus. Zentrales Kriterium für die Art der Spezialisierung ist die Art der Quantifizierung prädikatenlogischer Formeln in konjunktiver Normalform. Sogenannte Allaussagen (führende Allquantoren, z.B.: $\forall X, Y | ...$ oder $\forall X \exists Y | ...$) und Existenzaussagen (führende Existenzquantoren, z.B.: $\exists X, Y | ...$ oder $\exists X \forall Y | ...$) werden unterschieden.

Eine Allaussage kann nur dann durch eine Einfügung/Löschung verletzt werden, wenn das einzufügende/zu löschende Fakt mit einem negativen/positiven Literal der Integritätsbedingung unifiziert werden kann (inverse Polarität des Literals (positiv/negativ) und der Modifikationsart $(-/+)$). Eine solche Integritätsbedingung heißt **relevant** für eine gegebene Änderung (betroffen, 'affected'). Die Integritätsbedingung wird mit dem zu ändernden Fakt substituiert. Die so teilweise instanziierte Formel kann simplifiziert werden, indem das Auftreten des unifizierbaren Literals durch den zugehörigen Wahrheitswert ersetzt wird. Tritt dieses Literal positiv/negativ auf, so wird es durch den Wahrheitswert TRUE/FALSE ersetzt.

Beispiel 4.1.1 (Spezialisierung von Allaussagen)
Die Integritätsbedingung ic_1 aus dem Beispiel 2.1.17 stellt als prädikatenlogische Formel die Inklusionsbeziehung des Attributs 'mitarbeiter.abteil' zu 'abteilung.nr' dar.

- ic_1: $\forall X_1, X_2, X_3, X_4 \; \exists Y_2 \mid \neg mitarbeiter(X_1, X_2, X_3, X_4) \vee abteilung(X_2, Y_2)$
- *Für eine gegebene Einfügung $+mitarbeiter(Fritz, 10, 5000, Hugo)$ kann ic_1 nach den Vorgaben von Nicolas und Bernstein/Blaustein wie folgt spezialisiert werden:*

$$\begin{aligned} & \exists Y_2 \mid \neg mitarbeiter(Fritz, 10, 5000, Hugo) \vee abteilung(10, Y_2) \\ \Leftrightarrow\ & \exists Y_2 \mid \neg\ TRUE \vee abteilung(10, Y_2) \\ \Leftrightarrow\ & \exists Y_2 \mid abteilung(10, Y_2). \end{aligned}$$

Würde eine solche Spezialisierung bei Existenzaussagen durchgeführt, so hätte dies u.U. fehlerhafte Ergebnisse bei einer Integritätsprüfung zur Folge.

Beispiel 4.1.2 (Problem der Spezialisierung bei Existenzaussagen)
Sei ic eine Integritätsbedingung, mittels der die Existenz mindestens eines Mitarbeiter ohne einen vorgesetzten Manager kontrolliert wird. Gibt es keinen Vorgesetzten für einen Mitarbeiter, so gilt für die Bereichsvariable des 4. *Arguments des 'mitarbeiter'-Literals, daß* $X_4 = 0$.

$$\begin{aligned} & ic: \quad \exists X_1, X_2, X_3, X_4 \mid mitarbeiter(X_1, X_2, X_3, X_4) \ \wedge\ X_4 = 0 \\ & \text{gegebene Fakten: } \{mitarbeiter(Hugo, 13, 5000, 0), mitarbeiter(Erna, 13, 6000, 0)\} \end{aligned}$$

Angewendet auf die Faktenmenge ist 'ic' erfüllt und die Löschung $-mitarbeiter(Erna, ...)$ *verletzt die Existenzaussage nicht. Würde für 'ic' die gleiche Spezialisierung wie bei den Allaussagen durchgeführt, so würde die Anwendung der spezialisierten Bedingung als Ergebnis* FALSE *liefern und somit fälschlicherweise einen Integritätsfehler anzeigen.*

$$\begin{aligned} & \text{substituierte Formel } ic: \quad mitarbeiter(Erna, 13, 6000, 0) \ \wedge\ 0 = 0 \\ & \text{spezialisierte Formel } ic: \quad FALSE \wedge TRUE \end{aligned}$$

Für die Problematik der Spezialisierung von Integritätsbedingungen mit führendem Existenzquantor ('non-complacent assertions') sind insbesondere von Bernstein/Blaustein in [BB82] Lösungen entwickelt worden. Ihre Idee basiert auf der Generierung von Vortests ('partial test', 'quick test'), die weniger auswertungsaufwendig als die ursprüngliche Formel sind. Wird ein Vortest zu TRUE ausgewertet, so ist auch die Integritätsbedingung mit Sicherheit erfüllt. Wird der Vortest zu FALSE ausgewertet, so muß die Integritätsbedingung zudem ausgewertet werden, um entscheiden zu können, ob eine Konsistenzverletzung vorliegt oder nicht. Bei einer optimistischen Annahme, daß in der Praxis die korrekten Änderungen überwiegen, kann eine solche Strategie einen wesentlichen Performanzvorteil in sich bergen. Die Spezialisierung von Integritätsbedingungen, die keine Allaussagen sind, wird nicht weiter erörtert, da diese Problematik unabhängig vom zugrunde liegenden Kontext (Prädikatenlogik, Datalog, SQL) ist. Die bisherigen Optimierungsideen für Existenzaussagen können unmittelbar nach SQL übertragen werden.

Δ-Relationen und Transitionsregeln:

Um wie in den Beispielen die Spezialisierung nicht für ein konkretes Fakt während der Transaktion durchführen zu müssen, können die Integritätsbedingungen während des Schemaentwurfs für sogenannte **Δ-Relationen** (δ, 'delta relation', 'incremental', 'differential', 'internal events') spezialisiert werden. Für jede Relation und für jede Änderungsart wird eine Δ-Relation erzeugt (Einfügung: Δ^+, Löschung: Δ^-). Zum Zeitpunkt der Integritätsprüfung liegen in den Δ-Relationen die eingefügten bzw. gelöschten Fakten der Relationen vor. Die Integritätsbedingungen, die Allaussagen sind, werden derart transformiert, daß sie statt auf der manipulierten Relation je nach Polarität auf eine der

Δ^*-Relationen ($* \in \{^+,^-\}$) angewendet werden. Für die Prüfung von Allaussagen ist es ausreichend, derart spezialisierte Integritätsbedingungen anzuwenden.

Beispiel 4.1.3 (Spezialisierte Integritätsbedingungen)
Für die Integritätsbedingung aus Beispiel 4.1.1 kann die für Einfügungen in die 'mitarbeiter'-Relation spezialisierte Integritätsbedingung wie folgt modelliert werden

$$\Delta^+ ic_1\colon\ \forall X_1, X_2, X_3, X_4\ \exists Y_2 \mid \neg\Delta^+ mitarbeiter(X_1, X_2, X_3, X_4) \vee abteilung(X_2, Y_2).$$

Unabhängig von der Idee der Spezialisierung sind zwei alternative Prüfungszeitpunkte denkbar. Im optimistischen Fall werden erst die Änderungsanweisungen einer Transaktion ausgeführt, so daß zum Prüfungszeitpunkt bereits die neue Faktenmenge $\mathcal{F}^{new}$ vorliegt. Beim pessimistischen Ansatz werden die Integritätsbedingungen vor der Ausführung der Änderungsanweisungen geprüft. Da zu diesem Zeitpunkt nur der alte Zustand $\mathcal{F}^{old}$ vorliegt, muß, da Integritätsbedingungen auf dem neuen Zustand auszuwerten sind, die neue Faktenmenge $\mathcal{F}^{new}$ mittels sogenannter **Transitionsregeln** simuliert werden. $\mathcal{F}^{new}$ wird auf der Basis von $\mathcal{F}^{old}$ und den Δ^*-Relationen ermittelt ($* \in \{^+,^-\}$).

Beispiel 4.1.4 (Datalog-Integritätsbedingungen und Transitionsregeln)
Sei die Inklusionsbedingung ic_1 aus Beispiel 4.1.1 in Form einer Datalog-Integritätsbedingung gegeben. Zum Zeitpunkt der Änderungspropagierung liegt in den Relationen der alte Zustand vor, der neue Zustand wird simuliert.

- $ic_1 \leftarrow \neg h_1$ *WITH* $h_1 \leftarrow mitarbeiter(X_1, X_2, X_3, X_4) \wedge \neg h_2(X_2)$;
  $h_2(Y_1) \leftarrow abteilung(Y_1, Y_2)$.

- *Für Einfügungen in 'mitarbeiter' und Löschungen aus 'abteilung' können die spezialisierten Integritätsbedingungen $\Delta^+ ic_1, \Delta^- ic_1$ formuliert werden.*

  $\Delta^+ ic_1 \leftarrow \neg h_1$ *WITH* $h_1 \leftarrow \Delta^+ mitarbeiter(X_1, X_2, X_3, X_4) \wedge \neg h_2^{new}(X_2)$;
  $h_2^{new}(Y_1) \leftarrow abteilung^{new}(Y_1, Y_2)$;

  $\Delta^- ic_1 \leftarrow \neg h_1$ *WITH* $h_1 \leftarrow mitarbeiter^{new}(X_1, X_2, X_3, X_4) \wedge \neg\Delta^- h_2(X_2)$;
  $\Delta^- h_2(Y_1) \leftarrow \Delta^- abteilung(Y_1, Y_2)$.

- *Transitionsregeln zur Simulation von 'abteilung^{new}' basierend auf dem alten Zustand und den Δ-Relationen können wie folgt formuliert werden.*

  $abteilung^{new}(Y_1, Y_2) \leftarrow abteilung(Y_1, Y_2) \wedge \neg\, \Delta^- abteilung(Y_1, Y_2)$;
  $abteilung^{new}(Y_1, Y_2) \leftarrow \Delta^+ abteilung(Y_1, Y_2)$.

Propagierungsregeln (Δ-Regeln):

Die Idee der Spezialisierung auf der Basis von Δ-Relationen wurde Mitte der 80er Jahre u.a. von Lloyd et.al. ([LST86]) und Decker ([Dec86]) auf Deduktionsregeln mit dem Ziel der Änderungspropagierung übertragen. Die durch eine Basisfaktenänderung induzierten Änderungen von Sichten werden mittels **Δ-Regeln** (Propagierungsregeln) abgeleitet.

Für jedes Rumpfliteral einer Deduktionsregel werden zwei Δ^*-Regeln ($* \in \{^+, ^-\}$) generiert. Die Polarität des Rumpfliterals bestimmt die Polarität des Δ-Fakts des Kopfliterals. Eine Einfügung/Löschung bzgl. eines positiven Rumpfliterals wird als Einfügung/Löschung bzgl. des Kopfliterals propagiert. Eine Einfügung/Löschung bzgl. eines negativen Rumpfliterals hingegen wird als Löschung/Einfügung des Kopfliterals propagiert. Ein einzufügendes Fakt muß im neuen Zustand $\mathcal{F}^{new}$ ableitbar sein, und ein zu löschendes Fakt muß im alten Zustand $\mathcal{F}^{old}$ ableitbar gewesen sein, so daß die Seitenliterale einer Δ-Regel in Abhängigkeit der Polarität der propagierten Änderung bzgl. $\mathcal{F}^{new}$ oder $\mathcal{F}^{old}$ angewendet werden. Als **Seitenliteral (Residuum)** werden die Rumpfliterale einer Formel bezeichnet, die keine Δ-Literale sind. Je nach zugrunde liegendem Zustand wird entweder der neue $\mathcal{F}^{new}$ oder der alte Zustand $\mathcal{F}^{old}$ mittels Transitionsregeln simuliert.

Beispiel 4.1.5 (Datalog-Δ-Regeln)
Sei die Datalog-Regel für die abgeleitete Relation lower_mgr aus dem Beispiel 2.1.17 gegeben.

$$lower\_mgr(X_1, X_2, X_3) \leftarrow mita\_mgr(X_1, X_2, X_3) \wedge \neg h_1(X_3)$$
$$\textit{WITH}\ \ h_1(Y_1) \leftarrow mitarbeiter(Y_1, 13, Y_3, Y_4);$$

Spezialisierte Δ-Regeln können wie folgt modelliert werden:

$$\Delta^{\dagger} lower\_mgr(X_1, X_2, X_3) \leftarrow \Delta^{\dagger} mita\_mgr(X_1, X_2, X_3) \wedge \neg h_1^*(X_3)$$
$$\textit{WITH}\ \ h_1^*(Y_1) \leftarrow mitarbeiter^*(Y_1, 13, Y_3, Y_4);$$

$$mit\ \ (\dagger := {}^{+} \wedge * := {}^{new}) \ \vee \ (\dagger := {}^{-} \wedge * := {}^{old}).$$

$$\Delta^{\dagger} lower\_mgr(X_1, X_2, X_3) \leftarrow mita\_mgr^*(X_1, X_2, X_3) \wedge \neg \Delta^{\ddagger} h_1(X_3)$$
$$\textit{WITH}\ \ \Delta^{\ddagger} h_1(Y_1) \leftarrow \Delta^{\ddagger} mitarbeiter(Y_1, 13, Y_3, Y_4);$$

$$mit\ \ (\dagger := {}^{+} \wedge \ddagger := {}^{-} \wedge * := {}^{new}) \vee (\dagger := {}^{-} \wedge \ddagger := {}^{+} \wedge * := {}^{old}).$$

Bei der Definition der Datalog-Integritätsbedingungen (Abschnitt 2.1.1) wurde bereits darauf hingewiesen, daß sie parameterlose Deduktionsregeln mit dem Unterschied sind, daß ihre Ergebnisse (leere Menge/leeres Tupel) von der Integritätskomponente interpretiert werden und mit dem Festschreiben oder Zurückrollen der Änderungen reagiert wird. Gleiches gilt für die spezialisierten Integritätsbedingungen. Sie können im Rahmen eines Änderungspropagierungsprozesses analog zu den Δ-Regeln behandelt werden.

Skizze einer inkrementellen Verarbeitung im Datalog-Kontext:

Zur Schemaentwurfszeit können für eine gegebene Datalog-Datenbank $\mathcal{D}^{\mathcal{D}}$ die relevanten Δ-Basisrelationen zur Speicherung der Δ-Fakten von Basisfaktenänderungen sowie die spezialisierten Deduktionsregeln und Integritätsbedingungen generiert werden.

Die Ausführung einer Datalog-Transaktion, so wie sie in Abschnitt 2.1.3 definiert wurde (Definition 2.1.27), beinhaltet die Berechnung des Nettoeffekts der auszuführenden Transaktion $\mathcal{M}$, so daß zum Zeitpunkt der Propagierung bereits die Nettoeffektänderungen vorliegen. Zum Transaktionsende kann dann für eine gegebene Faktenmenge $\mathcal{F}^{old}$ und

dem Nettoeffekt einer Transaktion $\mathcal{M}$ ein Prozeß zur inkrementellen Änderungspropagierung und Integritätsprüfung durchgeführt werden. Für die Anwendung der Δ-Regeln und Δ-Integritätsbedingungen können die gleichen Verfahren wie bei der Anfrageauswertung angewendet werden, womit sowohl 'top down'-Auswertungsmethoden wie SLDNF-Resolution verwendet werden können wie auch 'bottom up'-Methoden zur Ausführung iterativer Fixpunktprozesse. Detailliert werden verschiedene Auswertungsverfahren in Abschnitt 4.2.2 diskutiert.

Für das nachfolgend definierte generische Verfahren wird ein iterativer Fixpunktprozeß zur Berechnung der induzierten Änderungen vorausgesetzt. Deduktionsregeln, Δ-Regeln, Integritätsbedingungen und spezialisierte Integritätsbedingungen werden gemäß ihren Abhängigkeiten Stratifikationsebenen zugeordnet. Stratum für Stratum werden die induzierten Änderungen propagiert und ihre Konsistenz wird unmittelbar geprüft. Für die Integritätsprüfung werden Δ-Integritätsbedingungen angewendet, die Ergebnisses der Integritätsprüfung werden interpretiert und entsprechend darauf reagiert. Ist die Konsistenz nicht gewährleistet, so wird die Transaktion zurückgerollt. Nur wenn kein Integritätsfehler vorliegt, werden die Basisfaktenänderungen und induzierten Änderungen für Basisrelationen und materialisierte Sichten ausgeführt und die Transaktion beendet.

Bei dem hier gewählten pessimistischen Ansatz werden Änderungspropagierung und Integritätsprüfung in einem Prozeß verzahnt ausgeführt und erst anschließend die Sichtenaktualisierung mit der Konsequenz, daß während des gesamten Prozesses der alte Zustand der Faktenmenge vorliegt und der neue simuliert wird. Bei einem optimistischen Ansatz können alle drei Teilprozesse während der Bearbeitung der DB-Objekte eines Stratums verzahnt ausgeführt werden. Zuerst können die induzierten Änderungen abgeleitet werden, dann die materialisierten Sichten des Stratums aktualisiert und die betroffenen Integritätsbedingungen geprüft werden. Beim Zurückrollen müssen dann nicht nur die Δ-Fakten gelöscht werden, auch alle Sichtenaktualisierungen und Basisfaktenänderungen müssen zurückgerollt werden.

4.1.2 Generisches Verfahren

Nachdem im vorangegangenen Abschnitt die zentralen Ideen inkrementeller Verfahren veranschaulicht worden sind, wird in diesem Abschnitt ein generisches Änderungspropagierungsverfahren für rekursive Regelmengen ohne Aggregate und ohne Duplikate im Datalog-Kontext definiert. Lösungsansätze für die Aggregatfunktionen und Duplikate werden in den Abschnitten 4.3.1, 4.3.2 vorgestellt. Nur die beiden Änderungsarten Einfügung und Löschung werden behandelt. Die für die Propagierung von Modifikationen erforderlichen Erweiterungen werden in Abschnitt 4.3.3 diskutiert. Die nachfolgende Definition eines generischen Verfahrens basiert auf der Methode von Griefahn aus [Grie97]. Die bei Griefahn ausführlich erörterten 'Magic Sets'-Transformationen von Δ-Regeln für eine effiziente Anwendung der Effektivitätstests wird im Rahmen dieser Arbeit nicht diskutiert. Ebenso wird ein alternierender Fixpunktprozeß zur Auswertung von Regelmengen, die nach der 'Magic Sets'-Transformation unstratifizierbar sind, nicht erörtert. Diese Probleme werden im weiteren als Aufgaben der Anfragekomponente betrachtet.

Sei $\mathcal{D}^{\mathcal{D}}$ eine gegebene Datalog-Datenbank mit dem Schema $\mathcal{S}^{\mathcal{D}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C} \rangle$ und $\mathcal{R} \cup \mathcal{C}$ sei eine Menge stratifizierbarer Datalog-Regeln. Das in Abschnitt 2.1.3 definierte Datalog-

Transaktionskonzept wird zugrunde gelegt und vorausgesetzt, daß die Änderungspropagierung und Integritätsprüfung vor der Ausführung der Basisfaktenänderungen und Sichtenaktualisierung zum Transaktionsende ausgeführt werden.

Eine Deduktionsregel ist relevant für (betroffen von) einer Änderung, wenn die Regel direkt oder indirekt von der manipulierten Relation abhängt. Bei Integritätsbedingungen müssen zudem die Polaritäten der Abhängigkeit und der Änderungsart berücksichtigt werden. Allaussagen ($IC \leftarrow \neg H$) können aufgrund des negierten Rumpfliterals H nur durch Einfügungen in H verletzt werden, die wiederum entweder durch Einfügungen in eine negativ von IC abhängige Relation oder durch Löschungen aus einer positiv von IC abhängigen Relation induziert werden. Existenzaussagen ($IC \leftarrow H$) können nur durch Löschungen aus H verletzt werden, die durch Löschungen aus einer positiv von IC abhängigen Relation oder durch Einfügungen in eine negativ von IC abhängigen Relation induziert werden.

Definition 4.1.6 (Relevante Datalog-Regel/-Integritätsbedingung)
Sei $*M$ eine Datalog-Änderungsanweisung ($* \in \{+, -\}$).

- *Eine Deduktionsregel $S \leftarrow L_1 \wedge ... \wedge L_n \quad (R \in \mathcal{R})$ ist relevant für $*M$, gdw. $rel(S) < rel(M)$.*
- *Eine Integritätsbedingung $IC \leftarrow \neg H$ oder $IC \leftarrow H \quad (IC \in \mathcal{C})$ ist relevant für $*M$, gdw. $(rel(IC) <_{-} rel(M) \ \wedge \ * = +) \quad \vee \quad (rel(IC) <_{+} rel(M) \ \wedge \ * = -)$.*

Um nicht sowohl den alten als auch den neuen Zustand einer Datenbank vollständig materialisieren zu müssen, kann der neue Zustand aller Relationen mittels sogenannter einfacher Transitionsregeln simuliert werden. Den Begriff der einfachen Transitionsregel verwendet Griefahn ([Grie97]) in Abgrenzung zu den in Abschnitt 4.2.4 diskutierten inkrementellen Transitionsregeln.

Definition 4.1.7 (Einfache Datalog-Transitionsregeln)
Sei $R \in \mathcal{B} \cup \mathcal{R}$ eine Relation in $\mathcal{D}^{\mathcal{D}}$. R^{old} bezeichne den alten Zustand von R und $\Delta^* R$ mit $* \in \{^+, ^-\}$ die zugehörigen Δ-Relationen. Zur Simulation des neuen Zustands R^{new} werden zwei Datalog-Transitionsregeln der Form

$$R^{new} \leftarrow R^{old} \wedge \neg \Delta^- R$$
$$R^{new} \leftarrow \Delta^+ R.$$

formuliert.

Wird Datalog wie bislang üblich ohne WITH-Klausel definiert, so gilt die in der Literatur allgemein verwendete Vorschrift ([Kue91], [Oli91], [BMM92], [Grie97] u..v.m.) der Generierung von Δ-Regeln für alle Literale, die im Regelrumpf auftreten. Das Rumpfliteral L_i einer Regel $R \in \mathcal{R}$ ($rel(L_i) \in rel_{body}(R)$), für das die Spezialisierung durchgeführt wird, wird durch das entsprechende Δ-Literal ersetzt. Da Einfügungen im neuen Zustand ableitbar sein müssen, werden die Residuen L_j ($rel(L_j) \in rel_{body}(R), \ j \neq i$) von R über dem neuen Zustand L_j^{new} ausgewertet, und da Löschungen im alten Zustand ableitbar

sind, werden die Residuen über dem alten Zustand L_j^{old} ausgewertet. Aufgrund der Voraussetzung, daß während der Propagierung der alte Zustand in den Relationen vorliegt, bezeichnet L_j^{new} die Transitionssichten zur Simulation des neuen Zustands und für den alten Zustand kann direkt auf die Relationen zugegriffen werden ($L_j^{old} := L_j$). Bei der nachfolgenden Definition wird vorausgesetzt, daß materialisierte Sichten wie virtuelle in dem Sinne behandelt werden, daß nicht auf die materialisierten Fakten zugegriffen wird, sondern analog zu den virtuellen Sichten immer die Ableitungsvorschrift angewendet wird. Optimierungsmöglichkeiten durch Zugriff auf die gespeicherten abgeleiteten Fakten und die daraus resultierenden Anforderungen an die Definition von Transitionssichten und spezialisierten Ableitungsvorschriften werden im SQL-Kontext in Kapitel 5 diskutiert.

Ist in Datalog-Regeln die **WITH-Klausel** zugelassen, dann wird für eine Regel R und jedes im Regelrumpf oder in einer Hilfsregel auftretende Literal ($rel(L) \in rel_{body}(R)$), das nicht das Literal einer Hilfsregel ist ($rel(L) \notin rel_{help}(R)$), eine für $\Delta^{\ddagger}L$ spezialisierte $\Delta^{\dagger}R$-Regel mit $\dagger, \ddagger \in \{^{+}, ^{-}\}$ erzeugt. Die Literale persistenter Relationen im Regelrumpf von (anderen) Hilfsregeln werden wie die Residuen des spezialisierten Regelrumpfs behandelt. Die Literale von Hilfsregeln können zwar grundsätzlich unverändert bleiben, für eine vereinfachte Darstellung bekommen sie jedoch wie alle anderen Residuen den Exponenten ^{new},^{old} zugeordnet. Diese Umbenennung wird nicht nur für die Rumpfliterale von Hilfsrelationen sondern auch für die Kopfliterale der Hilfsregeln durchgeführt. Anders als bei den '*new*'-, '*old*'-Literalen der persistenten Relationen beziehen sich diese Literale nicht auf Transitionsrelationen sondern auf die angepaßten Hilfsregel-Definitionen in der WITH-Klausel. Die Rumpfliterale der Hilfsregeln werden behandelt, als wären die Hilfsregeln in den Regelrumpf der persistenten Relation aufgefaltet. Da in SQL auch Hilfsregeln zulässig sind, wird diese erweiterte Generierungsvorschrift auch für SQL-Δ-Sichten angewendet.

Definition 4.1.8 (Datalog-Δ-Regel für ableitbare Fakten)
Sei eine Datalog-Regel $R := S \leftarrow L_1 \wedge ... \wedge L_i \wedge ... \wedge L_n$ *WITH* $H \leftarrow L_{h_1} \wedge ..... \wedge L_{h_n}$ *gegeben und sei* $rel(L_i) \in rel_{body}(R) \setminus rel_{help}(R)$*, dann ist jede Regel der Form*

$$\Delta^{\dagger}S \leftarrow L_1^* \wedge ... \wedge \Delta^{\ddagger}L_i \wedge ... \wedge L_n^* \qquad \textit{WITH} \quad H^* \leftarrow L_{h_1}^* \wedge ..... \wedge L_{h_n}^*$$

eine Δ-Regel (Propagierungsregel) von S zur Propagierung ableitbarer Fakten, wobei gilt $n \geq 1$, $1 \leq i \leq n$ und

$$\begin{array}{llll} \dagger := {}^{+}, & * := {}^{new} & \textit{falls} & (\ddagger = + \wedge S <_{+1} L_i) \vee (\ddagger = - \wedge S <_{-1} L_i) \\ \dagger := {}^{-}, & * := {}^{old} & \textit{falls} & (\ddagger = + \wedge S <_{-1} L_i) \vee (\ddagger = - \wedge S <_{+1} L_i) \ . \end{array}$$

Die Menge aller Δ-Regeln zu einer Deduktionsregel $R \in \mathcal{R}$ wird mit R_{Δ} bezeichnet und die Menge der Δ-Regeln zu allen Deduktionsregeln $\mathcal{R}$ einer DB $\mathcal{D}^{\mathcal{D}}$ mit $\mathcal{R}_{\Delta}$. $\mathcal{B}_{\Delta}$ bezeichnet die Menge aller Δ-Relationen von Basisrelationen. Die Unterscheidung verschiedener Arten propagierter Änderungen wie effektive, sichere, ableitbare oder potentielle Änderungen sowie die Definition ihrer Δ-Regeln werden in Abschnitt 4.2.1 erörtert.

Gemäß dem Relevanzbegriff wird eine allquantifizierte Integritätsbedingung nur in Abhängigkeit von der Polarität der Änderung und der des Auftretens des Rumpfliterals spezialisiert. Für existenzquantifizierte Integritätsbedingungen wird die Formel nicht spezialisiert, sondern nur über dem neuen Zustand ausgewertet.

Definition 4.1.9 (Spezialisierte Datalog-Integritätsbedingung)
Sei eine Integritätsbedingung mit dem Kopfliteral IC der DB $\mathcal{D}^{\mathcal{D}}$ gegeben.

- *Ist $IC \leftarrow \neg H$ eine Allaussage mit der (Hilfs-)Regel H, dann ist eine spezialisierte Integritätsbedingung ein Ausdruck der Form*

  $$\Delta IC \leftarrow \neg\Delta^{+}H,$$

  wobei $\Delta^{+}H$ eine Δ-Regel für die (Hilfs-)Regel H ist.

- *Ist $IC \leftarrow H$ eine Existenzaussage mit der Hilfsregel H, dann ist eine spezialisierte Integritätsbedingung ein Ausdruck der Form*

  $$\Delta IC \leftarrow H^{new}$$

  wobei H^{new} eine einfache Transitionsregel ist, wenn H eine persistente Relation ist. Bezeichnet H eine Hilfsrelation, deren Regel in der WITH-Klausel auftritt, dann wird H auf den neuen Zuständen der Relationen im Regelrumpf angewendet.

- *Alle Hilfsregeln einer optionalen WITH-Klausel von IC, die nicht die spezialisierte H-Regel sind, werden wie H^{new} transformiert.*

IC_{Δ} bezeichnet die Menge aller spezialisierten Integritätsbedingungen von $IC \in \mathcal{C}$ und $\mathcal{C}_{\Delta} := \bigcup_{i=1}^{n} IC_{i\Delta}$ die Menge der spezialisierten Integritätsbedingungen aller Integritätsbedingungen einer DB $\mathcal{D}^{\mathcal{D}}$.

Da die Δ-Regeln und spezialisierten Integritätsbedingungen zulässige Datalog-Regeln sind, können sie von der Anfragekomponente des DB-Systems angewendet werden. Die Menge aller Δ-Fakten kann mittels des $T_{\mathcal{R}}$-Operators abgeleitet werden.

Definition 4.1.10 (Menge induzierter Änderungen)
Sei $\mathcal{RC}_{\Delta} := \mathcal{R}_{\Delta} \cup \mathcal{C}_{\Delta}$ die Menge der Δ-Regeln für $\mathcal{R} \cup \mathcal{C}$ und $\mathcal{B}_{\Delta}$ die Menge der Δ-Relationen der Basisrelationen von $\mathcal{D}^{\mathcal{D}}$. $\mathcal{F}^{old}$ sei die Menge der gespeicherten Fakten und $\mathcal{F}^{B\Delta}$ die Menge der Δ-Fakten der Basisfaktenänderungen einer Transaktion $\mathcal{M}$.
Die Menge aller induzierten Änderungen (Δ-Fakten) $\mathcal{F}^{RC\Delta}$ ist durch die Anwendung des $T_{\mathcal{R}}$-Operators ableitbar.

$$\mathcal{F}^{RC\Delta} := T_{RC_{\Delta}}(\mathcal{F}^{old} \cup \mathcal{F}^{B\Delta})$$

wobei gilt $T_{RC_{\Delta}}(\mathcal{F}^{old} \cup \mathcal{F}^{B\Delta}) := \bigcup_{R \in RC_{\Delta}} T[R](\mathcal{F}^{old} \cup \mathcal{F}^{B\Delta})$.

Definition 4.1.11 (Konsistenter Datenbank-Zustand)
Sei $\mathcal{C}_{\Delta}$ die Menge der spezialisierten Integritätsbedingungen von $\mathcal{D}^{\mathcal{D}}$ und $\mathcal{F}^{B\Delta}$ die Menge der Δ-Fakten der Basisfaktenänderungen einer Transaktion $\mathcal{M}$. Wird $\mathcal{M}$ auf der Faktenmenge $\mathcal{F}^{old}$ ausgeführt, dann ist $\mathcal{F}^{RC\Delta}$ die Menge aller durch $\mathcal{M}$ induzierten Δ-Fakten und $\mathcal{F}^{new}$ der neue Zustand der Faktenmenge. ($* \in \{^{+},^{-}\}$)
Der Datenbank-Zustand $\mathcal{F}^{new}$ von $\mathcal{D}^{\mathcal{D}}$ heißt konsistent, gdw.

$$\forall\ \Delta^{*}IC \in \mathcal{C}_{\Delta} \mid eval(\Delta^{*}IC,\ \mathcal{F}^{old} \cup \mathcal{F}^{B\Delta}) = \mathit{TRUE}.$$

Für einen als iterativen Fixpunktprozeß auszuführenden Propagierungsprozeß mit Sichtenaktualisierung und Integritätsprüfung kann folgender Algorithmus formuliert werden. Die Schleifenbedingungen dienen der Kontrolle, daß nur die für eine gegebene Transaktion relevanten Δ-Sichten und -Integritätsbedingungen angewendet werden (änderungsgetriebene Propagierung). Ausführlich werden Aspekte einer **Relevanzanalyse** zur Laufzeit im Abschnitt 4.2.5 erörtert. Die Schleifenbedingungen dienen aber auch dem Erkennen eines vorzeitigen Endes, bevor das maximale Stratum bearbeitet ist, wenn keine weiteren Δ-Fakten propagiert werden und für die propagierten Δ-Fakten keine weiteren Sichten und Integritätsbedingungen zu verarbeiten sind (relevant sind).

Es werden zwei Faktenbasen unterschieden, die ursprüngliche Faktenmenge $\mathcal{F}^{old}$ und die bis zum aktuellen Stratum i abgeleiteten Δ-Fakten in den Δ-Relationen $\mathcal{F}^{RC\Delta^i}$. Die Δ-Regeln ($\mathcal{R}_\Delta$), -Integritätsbedingungen ($\mathcal{C}_\Delta$) und Transitionssichten werden über beiden Faktenmengen angewendet. Wenn alle Δ-Fakten ermittelt und ihre Konsistenz geprüft ist, werden sie auf den Sichten ausgeführt, so daß erst nach der Ausführung der Prozedur '*aktualisiere*' der neue Zustand $\mathcal{F}^{new}$ vorliegt. Während einer Iterationsrunde werden zuerst die relevanten Integritätsbedingungen des aktuellen Stratums i geprüft und anschließend werden die Δ-Fakten des nächsten Stratums $i+1$ propagiert. Bei der Auswertung der Δ-Regeln und -Integritätsbedingungen wird vorausgesetzt, daß die Δ-Fakten in den Δ-Relationen gespeichert werden. Würden nur virtuelle Δ-Sichten vorausgesetzt, dann müßte jedes Auftreten von $\mathcal{F}^{RC\Delta^i}$ im Algorithmus durch die Menge der Δ-Fakten der Basisfaktenänderungen $\mathcal{F}^{B\Delta}$ ersetzt werden.

Algorithmus 4.1.12 (Sichtenaktualisierung und Integritätsprüfung)
Seien $\mathcal{R}_\Delta$ und $\mathcal{C}_\Delta$ die Mengen der spezialisierten Regeln und Integritätsbedingungen von $\mathcal{D}^{\mathcal{P}}$ ($\mathcal{RC}_\Delta := \mathcal{R}_\Delta \cup \mathcal{C}_\Delta$). $\mathcal{RC}^i_\Delta \subseteq \mathcal{RC}_\Delta$ sei die Menge der Δ-Regeln und -Integritätsbedingungen des Stratums i und $\mathcal{RM}$ die Menge der materialisierten Sichten. Sei $* \in \{^+, ^-\}$ und 'interpret' eine Funktion, die das Ergebnis der Anwendung parameterloser Regeln auf den Wahrheitswert FALSE oder TRUE abbildet.
$\mathcal{F}^{B\Delta}$ sei die Menge der Δ-Fakten der Basisfaktenänderungen einer Transaktion $\mathcal{M}$, die auf der Faktenmenge $\mathcal{F}^{old}$ ausgeführt wird, dann ist $\mathcal{F}^{RC\Delta^i}$ die Menge der $\mathcal{B}_\Delta$-Fakten und aller durch sie induzierten Δ-Fakten bis zum aktuellen Stratum i.

BEGIN
  -- Initialisieren des Stratum-Index
  i:=0;
  -- Initialisieren der bearbeiteten Δs mit den Δs der Basisrelationen
  $\mathcal{F}^{RC\Delta^i}$:= $\mathcal{F}^{B\Delta}$;
  -- Anwenden aller relevanten Δ-Regeln des aktuellen Stratums i
  WHILE i $\leq$ maximalem $\lambda(\mathcal{RC}_\Delta)$ $\wedge$ $\exists$ relevante Δ^*R-Regeln im Stratum i
  LOOP
    pruefe (i, $\mathcal{C}_\Delta$, $\mathcal{F}^{old}$, $\mathcal{F}^{RC\Delta^i}$);
    i:=i+1; *-- Stratum-Index inkrementieren*
    propagiere (i, $\mathcal{R}_\Delta$, $\mathcal{F}^{old}$, $\mathcal{F}^{RC\Delta^i}$);
  END LOOP
  aktualisiere ($\mathcal{R}_\Delta$);
  lösche alle Δ-Fakten;
END;

mit den Prozeduren:

pruefe $(i,\ \mathcal{C}_\Delta,\ \mathcal{F}^{old},\ \mathcal{F}^{RC\Delta^i})$*:*
−− Prüfen der Konsistenz für alle relevanten spezialisierten Integritäts-
−− bedingungen des aktuellen Stratums i
IF $\exists\Delta^*IC \mid \Delta^*IC \in \mathcal{C}_\Delta \ \wedge\ \lambda(\Delta^*IC) = i \ \wedge\ interpret(\Delta^*IC) =$ *FALSE*
THEN ROLLBACK; END IF;

propagiere $(i,\ \mathcal{R}_\Delta,\ \mathcal{F}^{old},\ \mathcal{F}^{RC\Delta^i})$*:*
−− Propagieren der Änderungen für alle relevanten Δ-Regeln $\mathcal{RC}^i_\Delta$
−− des aktuellen Stratums i und speichern der Δ-Fakten in den Δ-Relationen
$\mathcal{F}^{RC\Delta^i} := \mathcal{F}^{RC\Delta} \cup T_{RC^i_\Delta}(\mathcal{F}^{old} \cup \mathcal{F}^{RC\Delta})$*;*

aktualisiere $(\mathcal{R}_\Delta)$*:*
−− Aktualisieren aller Basisrelationen und materialisierten Sichten,
−− für die Δ-Fakten vorliegen
FORALL $\Delta^*R \mid \Delta^*R \neq \emptyset \ \wedge\ R \in (\mathcal{RM} \cup \mathcal{B})$ *LOOP*
$*R$ *WHERE* Δ^*R;
END LOOP.

Um zu betonen, daß Δ-Regeln $\mathcal{R}_\Delta$ und spezialisierte Integritätsbedingungen $\mathcal{C}_\Delta$ auf die gleiche Art von einer Regelauswertungskomponente behandelt werden können, wird in dem Algorithmus die Regelmenge $\mathcal{R}_\Delta \cup \mathcal{C}_\Delta$ gemeinsam angewendet. Die Interpretation der Ergebnisse der Anwendung der spezialisierten Integritätsbedingungen wird anschließend in der *pruefe*-Prozedur von einer Funktion *interpret* durchgeführt, die das Ergebnis der Anwendung parameterloser Regeln auf TRUE abbildet, falls der Regelrumpf erfüllt ist, und anderenfalls auf FALSE.

4.2 Alternative Ansätze

4.2.1 Granularität induzierter Änderungen

Im Rahmen des einleitenden Beispiels werden sogenannte ableitbare Δ-Fakten propagiert. Vier Klassen induzierter Änderungen können unterschieden werden: potentielle ('potential'), sichere ('safe'), ableitbare ('derivable') und echte ('true'/'effective') Einfügungen und Löschungen. Diese verschiedenen Arten der Δ-Fakten unterscheiden sich bzgl. des Grades ihrer Exaktheit (Granularität, Qualität) und dem Auswertungsaufwand, der für ihre Ableitung erforderlich ist. Die Granularität der induzierten Änderung bestimmt auch die Definition der Δ-Regeln (Definition 4.2.2). Die Klassifizierung orientiert sich an dem Schema aus [Grie97], wobei Griefahn sich auf die drei Klassen der potentiellen, sicheren und effektiven Änderungen beschränkt.

Definition 4.2.1 (potentielle/sichere/ableitbare/echte Änderungen)
Sei $\mathcal{F}^{old}$ *der alte Zustand vor und* $\mathcal{F}^{new}$ *der neue Zustand nach Ausführung einer Transaktion* $\mathcal{M}$ *mit den induzierten Änderungen* $+M$ *und* $-M$*.*

1. $+M/-M$ *ist eine potentielle induzierte Einfügung/Löschung, gdw. nur die Variablenbindungen, die durch das geänderte Fakt im Rumpfliteral entstehen, an die Variablen des Kopfliterals weitergereicht werden.*

2. $+M$ *ist eine sichere induzierte Einfügung* *gdw.* $M \in \mathcal{F}^{new}$.
   $-M$ *ist eine sichere induzierte Löschung* *gdw.* $M \notin \mathcal{F}^{new}$.

3. $+M$ *ist eine ableitbare induzierte Einfügung* *gdw.* $M \in \mathcal{F}^{new}$.
   $-M$ *ist eine ableitbare induzierte Löschung* *gdw.* $M \in \mathcal{F}^{old}$.

4. $+M$ *ist eine echte induzierte Einfügung* *gdw.* $M \in \mathcal{F}^{new} \wedge M \notin \mathcal{F}^{old}$.
   $-M$ *ist eine echte induzierte Löschung* *gdw.* $M \in \mathcal{F}^{old} \wedge M \notin \mathcal{F}^{new}$.

Da bei der Propagierung potentieller Änderungen keine Residuen des Regelrumpfes ausgewertet werden, können Variablenbindungen bei der induzierten Änderung verlorengehen. So wird z.B. für die Regel $p(X,Y) \leftarrow q(X,Y,Z) \wedge r(Z)$ und die Einfügung $+r(3)$ die potentielle Änderung $+p(X,Y)$ abgeleitet. Sind alle Variablenbindungen bei einer potentiellen Änderung verlorengegangen, so kann zur Laufzeit auf der Basis dieser potentiellen Änderung nur noch ermittelt werden, welche Sichten und Integritätsbedingungen für die Änderung relevant sind (Abschnitt 4.2.5). Die gleiche Aussage kann auf der Basis eines zur Schemaentwurfszeit erstellten Abhängigkeitsgraphen getroffen werden.

Die Ableitung effektiver Änderungen zeichnet sich gegenüber der Propagierung ableitbarer Änderungen durch die Prüfung der Effektivität der Änderung aus (Effektivitätstest). Ein einzufügendes Fakt ($+M$) darf nicht bereits im alten Zustand vorgelegen haben ($M \notin \mathcal{F}^{old}$) und ein zu löschendes Fakt ($-M$) darf im neuen Zustand nicht über alternative Ableitungswege ableitbar bleiben ($M \notin \mathcal{F}^{new}$).

Definition 4.2.2 (Datalog-Δ-Regel für Änderungen)
Sei eine Datalog-Regel $S \leftarrow L_1 \wedge ... \wedge L_i \wedge ... \wedge L_n$ $(n \geq 1)$ *gegeben mit dem Rumpfliteral* L_i*, für das die Spezialisierung durchgeführt wird. Nach der Methode von Griefahn können die* Δ*-Regeln für die verschiedenen Typen von* Δ*-Fakten wie folgt definiert werden.*

- *Für potentielle induzierte Änderungen sind* Δ*-Regeln Ausdrücke der Form*

$$\Delta^{\dagger}S \leftarrow \Delta^{\ddagger}L_i.$$

- *Für sichere induzierte Einfügungen* $+S$*/Löschungen* $-S$ *sind* Δ*-Regeln Ausdrücke der Form*

$$\begin{aligned}\Delta^{+}S &\leftarrow L_1^{new} \wedge ... \wedge \Delta^{\ddagger}L_i \wedge ... \wedge L_n^{new}\\ \Delta^{-}S &\leftarrow \Delta^{\ddagger}L_i \wedge \neg S^{new}.\end{aligned}$$

- *Für ableitbare induzierte Änderungen sind* Δ*-Regeln Ausdrücke der Form*

$$\Delta^{\dagger}S \leftarrow L_1^{*} \wedge ... \wedge \Delta^{\ddagger}L_i \wedge ... \wedge L_n^{*}.$$

- *Für echte induzierte Änderungen sind* Δ*-Regeln Ausdrücke der Form*

$$\Delta^{\dagger}S \leftarrow L_1^{*} \wedge ... \wedge \Delta^{\ddagger}L_i \wedge ... \wedge L_n^{*} \wedge \neg S^{\star},$$

$$\begin{array}{llllllllll} wobei & \dagger := {}^{+}, & * := {}^{new} & \star := {}^{old} & falls & \ddagger & = & + & und & S < +1L_i \\ & & & & oder & \ddagger & = & - & und & S < -1L_i \\ & \dagger := {}^{-}, & * := {}^{old} & \star := {}^{new} & falls & \ddagger & = & + & und & S < -1L_i \\ & & & & oder & \ddagger & = & - & und & S < +1L_i. \end{array}$$

Die Δ-Regeln für potentielle Änderungen und für sichere Löschungen sind nicht bereichsbeschränkt. Bei potentiellen Änderungen ist die fehlende Bereichsbeschränkung unerheblich, da diese Regeln nicht ausgewertet werden. Die Ableitungsregel für sichere Löschungen muß soweit um Rumpfliterale $L_1^{old}, ..., L_n^{old}$ ergänzt werden, bis die Bereichsbeschränkung gewährleistet ist[1]. Ggf. entspricht die erweiterte Regel der Δ-Regel zur Ableitung effektiver Löschungen.

Dem Vorteil des wesentlich geringeren Aufwands bei der Ableitung potentieller, sicherer oder ableitbarer Folgeänderungen steht ggf. ein Mehraufwand bei der Integritätsprüfung, Sichtenaktualisierung und auch bei den weiteren Propagierungsschritten gegenüber. Bei ableitbaren und sicheren Änderungen wird u.U. eine Obermenge echter Einfügungen propagiert und bei den ableitbaren Fakten werden ggf. Fakten gelöscht, für die es alternative Ableitungswege gibt, bzw. bei sicheren Löschungen werden Fakten gelöscht, die im alten Zustand nicht vorlagen. Gehen bei der Propagierung potentieller Änderungen alle Variablenbindungen verloren, so müssen bei der Integritätsprüfung alle Fakten der Relation getestet bzw. die Sicht vollständig rematerialisiert werden. Ein Vorteil der Propagierung effektiver Änderungen ist zudem, daß Konfliktfälle nicht auftreten können, obwohl Mengen von Fakten propagiert werden. Weitere Nettoeffektberechnungen während eines Propagierungsprozesses sind nicht erforderlich (Abschnitt 2.1.3).

In der Literatur sind Propagierungsverfahren für unterschiedliche Kombinationen der Änderungsarten entwickelt worden.

- potentielle Änderungen: [LST86], [Kue91], [Grie97]
- sichere Änderungen: [Grie97]
- ableitbare Änderungen: [CW90], [CW91], [Kue91], [GMS93], [CW94], [GL95], [SBLC00]
- echte Änderungen: [Dec86], [Kue91], [Oli91], [BMM92], [UO92], [UO94], [TO95], [GrMa94], [Man94], [Grie97], [MT99], [MT00]
- sichere Einfügungen und echte Löschungen: [KSS87]
- sichere Einfügungen und potentielle Löschungen: [DW89]

Bei Propagierungsverfahren, bei denen Duplikate zugelassen sind ([GMS93]: Counting, [GL95], [BDDF+98], [SBLC00]), ist ein Effektivitätstest aufgrund der Multimengensemantik nicht sinnvoll. Statt die Existenz alternativer Ableitungswege zu testen, werden sie bei Gupta et.al. gezählt. Bei Griffin et.al. werden alle ableitbaren Fakten gespeichert (Abschnitt 4.3.2).

Da zum Ende einer Datalog-Transaktion ggf. eine Menge von Änderungen bzgl. verschiedener Basisrelationen zu propagieren ist, können bei der Anwendung der Δ-Regeln

[1] Ausführlich wird diese Problematik in [Grie97] 7.2.4 diskutiert.

(Definition 4.2.2) Änderungen, die durch verschiedene Basisfaktenänderungen induziert werden, wiederholt abgeleitet werden. Bei Einfügungen z.B. ist der Grund für diese Ineffizienz, daß die Seitenliterale jeweils im neuen Zustand (inkl. aller neuen Fakten) ausgewertet werden. In [GMS93] wird dieses Redundanzproblem syntaktisch dadurch gelöst, daß alle Seitenliterale links vom Δ-Rumpfliteral für den neuen und alle rechts davon für den alten Datenbank-Zustand ausgewertet werden. In [Man94] werden die Seitenliterale immer über den alten Zustand ausgewertet und eine zusätzliche Regel generiert, mittels der die Folgeänderungen abgeleitet werden, die nur auf den neuen Fakten (Δs) basieren. In [Oli91], [UO92], [UO94], [TO95] werden für eine Regel mit k Rumpfliteralen 2^k optimierte Transitionsregeln je Einfügung/Löschung (Modifikation: $2 * 2^k + 2^{2k}$) generiert. Es wird eine Regel generiert, die die unveränderte Faktenmenge ableitet. In jeder weiteren generierten Regel werden sukzessiv die Rumpfliterale durch die Δ-Atome ersetzt, bis zum Schluß eine Regel erzeugt wird, die ausschließlich auf Δ-Atomen ausgewertet wird. Diese Strategie hat ein exponentielles Wachstum der Transitionsregelmenge zur Folge.

Beispiel 4.2.3
Sei eine Regel $s(X,Y) \leftarrow p(X,Z) \wedge q(Z,Y)$ *gegeben, dann können gemäß den verschiedenen Verfahren folgende Δ-/Transitionsregeln für Einfügungen modelliert werden.*

Generisches Verfahren:
$$\begin{aligned}\Delta^+s(X,Y) &\leftarrow \Delta^+p(X,Z) \wedge q^{new}(Z,Y)\\ \Delta^+s(X,Y) &\leftarrow p^{new}(X,Z) \wedge \Delta^+q(Z,Y)\end{aligned}$$

[GMS93]:
$$\begin{aligned}\Delta^+s(X,Y) &\leftarrow \Delta^+p(X,Z) \wedge q^{o}(Z,Y)\\ \Delta^+s(X,Y) &\leftarrow p^{new}(X,Z) \wedge \Delta^+q(Z,Y)\end{aligned}$$

[Man94]:
$$\begin{aligned}\Delta^+s(X,Y) &\leftarrow \Delta^+p(X,Z) \wedge q^{old}(Z,Y);\\ \Delta^+s(X,Y) &\leftarrow p^{old}(X,Z) \wedge \Delta^+q(Z,Y)\\ \Delta^+s(X,Y) &\leftarrow \Delta^+p(X,Z) \wedge \Delta^+q(Z,Y)\end{aligned}$$

[Oli91], [UO92]:
$$\begin{aligned}s^{new}(X,Y) \leftarrow\ & (p^{old}(X,Z) \wedge \neg\Delta^-p(X,Z)) \wedge\\ & (q^{new}(Z,Y) \wedge \neg\Delta^-q(Z,Y))\\ s^{new}(X,Y) \leftarrow\ & (p^{old}(X,Z) \wedge \neg\Delta^-p(X,Z)) \wedge \Delta^+q(Z,Y)\\ s^{new}(X,Y) \leftarrow\ & \Delta^+p(X,Z) \wedge (q^{new}(Z,Y) \wedge \neg\Delta^-q(Z,Y))\\ s^{new}(X,Y) \leftarrow\ & \Delta^+p(X,Z) \wedge \Delta^+q(Z,Y).\end{aligned}$$

Auf diese Problematik wird mit dem Ziel weiterer Effizienzsteigerungen für den Fall einer konkreten Implementierung hingewiesen. Damit das Verfahren mit einer überschaubaren Regelmenge dargestellt werden kann, wird diese Ineffizienz akzeptiert. Eine Anpassung der im weiteren definierten Regelmengen ist trivial.

4.2.2 Auswertungsstrategien

Analog zur Problematik der Anfrageauswertung werden auch bei der Propagierung im Datalog-Kontext die beiden Basisstrategien 'top down' und 'bottom up' unterschieden.

- **'top down':** Induzierte Änderungen werden abgeleitet, indem Anfragen nach neuen Δ-Fakten an das DB-System gestellt werden. Für die Integritätsprüfung z.B. kann eine Integritätsbedingung modelliert werden, in deren Regelrumpf die Literale aller

spezialisierten Integritätsbedingungen konjunktiv verknüpft werden. Zu Prüfungszwecken muß dann nur diese Regel angewendet werden.

- **'bottom up':** Der Propagierungsprozeß wird als änderungsgetrieben angesehen. Basisfaktenänderungen induzieren bei den unmittelbar abhängigen Sichten Änderungen, die ihrerseits bei den von ihren direkt abhängigen Sichten Änderungen induzieren. Ein Fixpunktprozeß wird ausgeführt bis alle Änderungen ermittelt sind.

Von Kowalski et.al. wurde in [KSS87] zuerst vorgeschlagen, Integritätsbedingungen als Regeln zu definieren und sie mittels SLDNF-Resolution 'top down' auszuwerten. Küchenhoff bezeichnet in [Kue91] seine Auswertungskomponenten als konkrete Implementierung der von Kowalski et.al. vorgeschlagenen Lösung. Ebenfalls eine SLDNF-Resolution legen [BDM88] (Prolog-Implementierung), [Oli91], [UO92], [MT99], [MT00] zugrunde. Von Vorteil bei einem 'top down'-Ansatz ist die einfache Auswertungssteuerung. Im einfachsten Fall wie bei obiger Integritätsregel, in deren Rumpf die Literale aller spezialisierten Integritätsbedingungen auftreten, reicht es für die Integritätsprüfung aus, nur eine Formel anzuwenden. Ein grundlegender Nachteil ist das Terminierungsproblem, das bei SLDNF-Resolution für rekursive Regeln auftreten kann.

Andere Autoren ([Dec86], [LST86], [KSS87], [BDM88], [CW91], [QW91], [GMS93] [GL95], [Grie97], [SBLC00], [MT99], [MT00]) haben für ihre Verfahren insbesondere zur Sichtenaktualisierung einen 'bottom up'-Propagierungsprozeß entwickelt. Für eine 'bottom up'-Propagierung ist wie beim generischen Verfahren eine explizite Ausführungssteuerung erforderlich, die gewährleistet, daß Stratum für Stratum die Δ-Regeln und -Integritätsbedingungen angewendet werden, bis ein Fixpunkt erreicht ist. Die Δ-Regeln und spezialisierten Integritätsbedingungen werden ebenso wie die Deduktionsregeln der Sichten im Rahmen einer 'bottom up'-Anfrageauswertung (Abschnitt 2.1.2) Stratifikationsebenen zugeordnet.

Werden die Δ-Fakten gespeichert, so können aufgrund der stratumbezogenen Ausführung bei 'bottom up'-Prozessen wiederholte Ableitungen vermieden werden, da bei Zugriffen von höheren Stratifikationsebenen die Fakten in den Δ-Relationen bereits vorliegen. Ist bei einer 'top down'-Auswertungsstrategie die Anfragekomponente nicht in der Lage, wiederholte Zugriffe auf die gleichen Zwischenergebnisse (referenzierte Sichten) zu erkennen und diese Zwischenergebnisse zu speichern, sind wiederholte Ableitungen und ggf. längere Ausführungszeiten die Konsequenz. Wird diese Ausführungsoptimierung nicht der Anfragekomponente überlassen, so besteht die Schwierigkeit, während eines Propagierungsprozesses zu erkennen, welche Δ-Relationen bereits bearbeitet wurden, und welche leer sind, weil keine Δ-Fakten propagiert werden konnten, bzw. welche leer sind, weil die relevanten Δ-Regeln noch nicht angewendet wurden. Eine einfache Lösung stellt die Definition einer totalen Ausführungsordnung für alle Δ-Regeln und -Integritätsbedingungen dar. Eine solche Lösung würde zwar dem Grundgedanken einer mengenorientierten Propagierung widersprechen, aber korrekte Ergebnisse liefern.

Wie beim generischen Verfahren deutlich wurde, kann die Ausführungskontrolle für den Fixpunktprozeß (Schleifenbedingungen) sehr leicht um die Bedingungen erweitert werden, die gewährleisten, daß in Abhängigkeit von den aktuellen Basisfaktenänderungen der Transaktion nur die relevanten Δ-Regeln und -Integritätsbedingungen angewendet werden. Mittels einer einfachen 'top down'-Auswertung, wie sie oben für die Integritätsregel

mit den Literalen aller Δ-Integritätsbedingungen skizziert wurde, ist eine Beschränkung nur auf die relevanten Integritätsbedingungen nicht unmittelbar möglich. Die Durchführung von Relevanzanalysen zur Laufzeit können nur durch zusätzliche Ausführungskontrollen erreicht werden, die ähnlich aufwendig sind wie bei einer 'bottom up'-Steuerung.

In der Literatur können im wesentlichen zwei Implementierungsstrategien für 'bottom up'-Propagierungsverfahren unterschieden werden.

- **passive Implementierung:** Die spezialisierten Propagierungsvorschriften zur Ableitung induzierter Änderungen werden wie beim generischen Verfahren als Deduktionsregeln implementiert ([GMS93], [Man94], [SBLC00]) oder als Ausdrücke der zugrunde liegenden Algebra ([Dec86], [LST86], [GL95], [CGLMT96]). Die induzierten Änderungen werden mittels der DBS-internen Deduktions- bzw. Anfragekomponente abgeleitet. Die Steuerung der 'bottom up'-Ausführungsreihenfolge der Δ-Regeln/ -Integritätsbedingungen erfolgt prozedural.

- **aktive Implementierung:** Die spezialisierten Propagierungsvorschriften zur Ableitung induzierter Änderungen werden wie bei SQL-basierten Verfahren in Kapitel 5 als Trigger realisiert (Starburst: [CW90], [CW91], [CW94], Chimera: [CF97], [GrMa94], ActLog: [Grie97]). In den Verfahren werden sogenannte Propagierungstrigger für die Ableitung von Änderungen ausgeführt. Als Aktion leiten sie wiederum induzierte Änderungen ab, wodurch Propagierungstrigger der nachfolgenden Strata aktiviert werden. Die Triggerkomponente übernimmt (weitgehend) die Steuerung der Ausführung des iterierten Fixpunktprozesses für die Änderungspropagierung. Dadurch, daß Trigger u.a. für Basisfaktenänderungen feuern, ist die Realisierung eines änderungsgetriebenen Prozesses unmittelbar möglich.

Vor- und Nachteile von 'top down'- und 'bottom up'-Auswertungsstrategien werden u.a. bei [Ull89], [Bid91] und [CGH94] diskutiert und auch verschiedene Optimierungsansätze vorgestellt. Die Schwierigkeiten und Vorteile von triggerbasierten und deduktiven Implementierungen werden im Zusammenhang mit der Entwicklung eines Implementierungsansatzes für SQL-basierte Verfahren detailliert in Abschnitt 5.1.2 diskutiert. Andere triggerbasierte Implementierungsansätze werden in Abschnitt 5.7.2 erörtert.

4.2.3 Implementierung der Δ-Mengen

Für die Verwendung der Δ-Fakten sind drei Informationen relevant: die Art der Änderung, die betroffene Relation und die Ausprägung des geänderten Fakts. Die in Definition 4.1.8 gewählte Notation der Δ^+/Δ^--Relationen wird in modifizierter Form vielfach verwendet, so z.B. bei [Man94]: R^+, R^-; [Oli91], [UO92], [TO95], [MT00]: $\iota R, \delta R$; [QW91], [GL95], [CGLMT96]: $\triangle R, \bigtriangledown R$. Bei dieser Darstellung wird für jede Relation und jede (relevante) Änderungsart eine Δ-Relation generiert. Konsequenz der hier verwendeten Notation ist, daß sich die Menge der Relationen verdreifacht bzw. vervierfacht, wenn auch Modifikationen propagiert werden. Bei großen DB-Schemata kann dies einen beachtlichen Verwaltungsaufwand für das DB-System bedeuten. Eine alternative Darstellungsform faßt alle Änderungen in einer Menge zusammen ([GMS93], [BDDF+98]), so daß sich die Menge der Relationen nur verdoppelt. Ein zusätzliches Attribut gibt Auskunft über die Änderungsart. Eine weitere Darstellungsform ist die als unäre Relationen mit den geänderten Fakten als Argument wie bei [Kue91]: deltaplus(R), deltaminus(R); [BMM92]:

to-be-removed(R), to-be-inserted(R) oder [GMS93]: $\Delta(R)$. Da die Entscheidung für eine Darstellungsform die Laufzeit-Performanz beeinflussen kann, kann sie nur empirisch und in Abhängigkeit vom gegebenen DB-System und DB-Schema getroffen werden. Die erforderlichen Anpassungen an einen Propagierungsalgorithmus sind trivial.

Die Δ-Mengen von Basisrelationen werden als Basisrelationen implementiert. Anderenfalls müßten die alten und neuen Faktenmengen der Basisrelationen vollständig gespeichert vorliegen, damit die Δ-Fakten dann als Differenzen der beiden Mengen berechnet werden könnten. Die Δ-Mengen von Sichten können entweder virtuell oder gespeichert vorliegen. Ist die Regeldarstellung gewählt worden, so sind Lösungen möglich, bei denen entweder alle Δ-Sichten materialisiert bzw. virtuell definiert sind oder für jede Δ-Sicht individuell der Sichttyp festgelegt wird. Entscheidendes Kriterium ist der Konflikt zwischen den Kosten für wiederholte Ableitungen und den Kosten für die Speicherung der Δ-Fakten auf einem externen Speichermedium.

Bei den triggerbasierten Implementierungen der Verfahren von Ceri/Widom (Starburst: [CW90], [CW91], [CW94]) und Griefahn (ActLog: [Grie97]) wird auf die Realisierung von Δ-Mengen verzichtet, indem bereits existierende Systemkomponenten als Δ-Relationen verwendet werden. Da die Propagierungstrigger durch die Ausführung von Änderungen aktiviert werden, stehen in den Triggern alle relevanten Informationen zur Verfügung, die auch von den Δ-Fakten bereitgestellt werden. Starburst-Trigger feuern mengenorientiert für die Ausführung einer DML-Anweisung. Für den Zugriff auf die einzelnen manipulierten Fakten im Aktionsteil wird von der Triggerkomponente für die Ausführung jeder Änderungsanweisung die Relation INSERTED bzw. DELETED gepflegt. Für die Deduktion der induzierten Änderung im Triggeraktionsteil kann dann auf diese systemintern verwalteten 'Δ-Mengen' zugegriffen werden. ActLog-Trigger hingegen feuern tupelorientiert, und die Variablenbindungen mit Werten der feuernden DML-Anweisung werden unmittelbar auf die Aktionen im Triggeraktionsteil übertragen. Bei beiden Implementierungen muß vorausgesetzt werden, daß alle Sichten materialisiert sind, und Trigger auch bzgl. Sichten definierbar sind. Die induzierten Änderungen dürfen nicht als 'view updates' behandelt werden, die es erfordern würden, daß sie gemäß der Sichtendefinition in Basisfaktenänderungen transformiert und ausgeführt werden.

4.2.4 Transitionsregeln

Bei der Änderungspropagierung wird für Residuenauswertungen und Effektivitätstests auf unterschiedliche DB-Zustände zugegriffen, so müssen z.B. einzufügende Fakten im neuen Zustand ableitbar sein und zu löschende im alten ableitbar gewesen sein. Für effektive Änderungen muß geprüft werden, daß für Löschungen nicht weiterhin alternative Ableitungswege existieren und für Einfügungen, daß sie nicht bereits im alten Zustand existiert haben. Da es im allgemeinen nicht durchführbar ist, beide Faktenmengen $\mathcal{F}^{old}, \mathcal{F}^{new}$ einer Datenbank $\mathcal{D}^{\mathcal{D}}$ vollständig zu speichern, kann eine der Faktenmengen mittels Transitionsregeln (Olivé et.al.: 'transition rules') simuliert werden. Dieser Lösungsansatz ist in vielen Verfahren zu finden wie z.B. in [BDM88], [Kue91], [Oli91], [QW91], [UO92], [GMS93], [Grie97], [MT00].

Die Entscheidung, welcher Zustand simuliert wird, wird durch die beiden Zeitpunkte der Ausführung der Basisfaktenänderung und der Propagierung bestimmt. Werden zu-

erst die Basisfaktenänderungen ausgeführt und anschließend die Propagierung ([CW90], [CW91], [CW94]), dann muß der alte Zustand simuliert werden (optimistischer Ansatz). Findet die Propagierung wie bei [BDDF+98], [SBLC00], [CGLMT96] zu einem beliebigen Zeitpunkt nach Beendigung der Transaktion statt, muß ebenfalls der alte Zustand simuliert werden. Werden Propagierung und Prüfung vor einer Ausführung der Basisfaktenänderungen durchgeführt (pessimistischer Ansatz), so muß der neue Zustand simuliert werden ([BMM92], [Man94], [GMS93], [GL95], [QW91]).

In Abschnitt 4.1.2 sind sogenannte einfache Transitionsregeln definiert worden (Definition 4.1.7, [Grie97]: 'naive transition rules'). Obwohl sie allgemein in inkrementellen Verfahren verwendet werden, kann ihre Anwendung zu unstratifizierbaren Δ-Regeln führen. Bei rekursiven Δ^--Regeln für effektive induzierte Löschungen treten aufgrund des Effektivitätstests negative rekursive Abhängigkeiten auf ($p <_- p$). Dieses **Stratifikationsproblem** wird bereits in [Oli91] und [Man94] analysiert. Manthey verweist zur Lösung des Problems auf die Eigenschaft der lokalen Stratifizierbarkeit[2]. Olivé zeigt, daß die Programme semi-strikt[3] sind, und zur Lösung eine weitere Regel negiert in den Zyklus eingefügt werden muß.

Griefahn widerlegt in ihren Ausführungen ([Grie97] 7.3.1) anhand eines Beispiels die These von Olivé und entwickelt eine Lösung basierend auf direkten und indirekten Transitionsregeln, auch inkrementelle Transitionsregeln genannt ('incremental transition rules'). Durch die Verwendung indirekter Transitionsregeln für Sichten wird ausgeschlossen, daß die Δ-Regeln für die Propagierung effektiver Änderungen von stratifizierbaren, rekursiven Deduktionsregeln unstratifizierbar werden.

Definition 4.2.4 (Direkte und indirekte Transitionsregeln)
Sei $\mathcal{R}$ die stratifizierbare Regelmenge einer Datalog-Datenbank $\mathcal{D}^{\mathcal{D}}$. Die Exponenten ^{old},^{new} bezeichnen den alten/neuen Zustand einer Relation und Δ^+, Δ^- die Δ-Relationen für einzufügende und zu löschende Fakten.

- *Ist B eine Basisrelation ($B \in \mathcal{B}$), dann ist eine direkte Transitionsregel zur Simulation des neuen Zustands B^{new} ein Ausdruck der Form*

$$B^{new} \leftarrow B^{old} \wedge \neg\Delta^- B$$
$$B^{new} \leftarrow \Delta^+ B.$$

- *Ist eine Deduktionsregel $S \leftarrow L_1 \wedge \ldots \wedge L_n$ aus $\mathcal{R}$ gegeben ($1 \leq i \leq n$), dann ist eine indirekte Transitionsregel zur Simulation des neuen Zustands S^{new} ein Ausdruck der Form*

$$S^{new} \leftarrow \theta L_1^{new} \wedge \ldots \wedge \theta L_n^{new}$$

wobei θL_i^{new} eine direkte Transitionsregel ist, gdw. $rel(L_i) \in rel_{edb}(\mathcal{R})$,
oder θL_i^{new} eine indirekte Transitionsregel ist, gdw. $rel(L_i) \in rel_{idb}(\mathcal{R})$.

[2]Die Stratifizierbarkeit basiert nicht nur auf den Eigenschaften der Regelmengen sondern auch auf den grundinstanziierten Atomen.

[3]Semi-strikte Regelmengen erlauben eine gerade Anzahl von Negationen in Zyklen.

Das Stratifikationsproblem tritt bei der Verwendung einfacher Transitionsregeln nicht auf, wenn wie bei [CW91], [QW91], [GMS93], [GL95], [MT99], [MT00], [SBLC00] rekursive Sichten nicht zugelassen sind. Da bei potentiellen und ableitbaren Löschungen kein Effektivitätstest durchgeführt wird, tritt das Problem dort ebenfalls nicht auf. Nachteilig ist jedoch, daß eine Obermenge der effektiv zu löschenden Fakten propagiert wird. Manthey zeigt in [Man94], daß Effektivitätstests für die Ableitung effektiver Änderungen nur dann erforderlich sind, wenn eine Projektion durchgeführt wird. Um das Stratifikationsproblem zu umgehen und zugleich nur effektive Löschungen für das nächste Stratum abzuleiten, wird bei [CW94] (Re-Insert-Algorithmus) und [GMS93] (DRed-Algorithmus ('delete and rederive')) die Ableitung effektiver Löschungen in zwei unabhängigen Schritten durchgeführt. Im ersten Schritt werden alle ableitbaren Löschungen ermittelt und ausgeführt. Anschließend werden in einem zweiten Schritt die gelöschten Fakten, für die alternative Ableitungswege existieren, erneut abgeleitet und wieder eingefügt. Der Auswertungsaufwand bei der 'Wiederableitung' von Löschungen ist vergleichbar mit dem der Effektivitätstests. Bei der Re-Insert-Strategie stellt jedoch der Aufwand für die Ausführung zweier insgesamt effektloser Änderungsanweisungen (teure Speicherzugriffe) einen ggf. beachtlichen zeitlichen Mehraufwand dar.

4.2.5 Relevanzanalyse

Bereits bei der Einführung in die Grundlagen inkrementeller Verfahren wird ein grundlegendes Verständnis der Relevanzanalyse vermittelt und eine einfache Definition des Relevanzbegriffs im Datalog-Kontext gegeben (Definition 4.1.6). Ziel einer Relevanzanalyse ist, daß möglichst nur die von einer Änderung betroffenen Regeln und Integritätsbedingungen angewendet werden. Die Qualität einer Relevanzanalyse beeinflußt die Ausführungszeit eines Propagierungsprozesses entscheidend. Die Entscheidung der Relevanz für eine Änderung wird aufgrund der syntaktischen Strukturen einer Menge von Regeln und Integritätsbedingungen getroffen. Zur Schemaentwurfszeit können die Mengen von Regeln und Integritätsbedingungen zu diesem Zweck analysiert werden. Die Ergebnisse solcher syntaktischen Analysen können z.B. in Form von **Abhängigkeitsgraphen** für Relationen und Attribute (Abschnitt 2.1.1, [Grie97]: 'dependency graph'), von 'relevant sets' ([LL96]) oder von 'parse trees' ([CW90]) dargestellt werden.

Die gesammelten Informationen können sowohl im Rahmen des Schemaentwurfs als auch zur Laufzeit während der Propagierung verwendet werden. Während des Schemaentwurfs kann anhand dieser Informationen z.B. entschieden werden, welche spezialisierten Regeln und Integritätsbedingungen für ein DB-Schema $\mathcal{S}^{\mathcal{D}}$ relevant sind, oder, für welche Basisrelationen eine Δ-Basisrelation erzeugt werden muß. Für Basisrelationen, von denen weder direkt noch indirekt Regeln oder Integritätsbedingungen abhängen, müssen z.B. keine Δ-Fakten gespeichert werden. Auf diese Weise werden keine unnötigen DB-Objekte erzeugt. Während eines Propagierungsprozesses kann dann anhand der in den Abhängigkeitsgraphen vorliegenden Informationen z.B. entschieden werden, ob ein Fakt propagiert werden muß, oder, ob der Prozeß beendet werden kann, weil keine neuen Δ-Fakten abgeleitet werden. Ziel ist es, unnötige Verarbeitungsschritte zu vermeiden, um die Laufzeit zu minimieren.

Für Einfügungen und Löschungen kann die Relevanz i.d.R. ohne Zugriff auf die Faktenmenge nur auf der Basis der Informationen der Abhängigkeitsgraphen entschieden

werden (statische Analyse). Bei Modifikationen kann die Relevanz attributbezogen entschieden werden, so daß zur Laufzeit ggf. konkrete Wertänderungen einzelner Attribute berücksichtigt werden (dynamische Analyse). In Abschnitt 4.3.3 werden die Besonderheiten der Relevanzanalyse für Modifikationen erörtert.

Ausführliche Diskussionen der Kriterien und Strategien von Relevanzanalysen sind u.a. bei [Elkan90], [LS93], [LL96] zu finden. Die hier vorgestellte Relevanzanalyse gestaltet sich aufgrund der beschränkten Syntax von Datalog sehr intuitiv. Ceri/Widom haben in [CW90] eine ähnlich leistungsstarke Relevanzanalyse ('derivation of invalidating operations') für eine sehr umfassende SQL2-Syntax formuliert. Entsprechend komplex gestalten sich ihre Analysevorschriften, und die bestehenden Analogien zu den in Datalog bekannten Analyseverfahren werden nur sehr schwer erkennbar.

4.2.6 Implementierungen in kommerziellen Systemen

In diesem Abschnitt werden die Integritäts- und Sichtkonzepte der kommerziellen DB-Systeme der beiden führenden DBS-Hersteller Oracle (O 8.1.6 [Ora99][4]) und IBM (DB2 V 7.x) vorgestellt. Den Ausführungen liegen die in Abschnitt 2.2.3 definierten SQL3-Integritätsbedingungen zugrunde und die Abweichungen bei beiden Implementierungen werden gegenüber den Vorgaben des SQL3-Standards aufgezeigt. Gegenüber den bisherigen Erörterungen in diesem Kapitel ist zu beachten, daß SQL-Datenbanken auf einem von Datalog stark abweichenden Transaktionskonzept basieren.

Bei beiden DB-Systemen ist das **SQL3-Integritätskonzept** funktional sehr eingeschränkt implementiert. ASSERTIONs sind nicht definierbar, und in den CHECK-Bedingungen von CONSTRAINTs sind keine Anfragen auf Basisrelationen oder Sichten zugelassen. Diese Restriktionen zusammen mit der SQL3-konformen Eigenschaft, daß CHECK-CONSTRAINTS für Sichten nicht definierbar sind, schließen Integritätsbedingungen für Sichten aus, so daß eine inkrementelle Änderungspropagierung nicht erforderlich ist. Die Integritätsbedingungen sind zu den beiden Zeitpunkten IMMEDIATE und DEFERRED prüfbar, und somit ist das Konzept sehr gut an das SQL-Transaktionskonzept angepaßt.

Oracle stellt seit der Version 8.1 mit den **materialisierten Sichten** (CREATE MATERIALIZED VIEW) ein gegenüber dem seit 1992 bekannten Konzept der periodisch aktualisierbaren Snapshots (CREATE SNAPSHOT) wesentlich erweitertes Konzept zur Sichtenmaterialisierung zur Verfügung. Bello et.al. erläutern in [BDDF+98] ausführlich den Prozeß der inkrementellen Änderungspropagierung mit dem Ziel der Sichtenaktualisierung.

Eine materialisierte Sicht wird mittels einer Basisrelation simuliert. Da Oracle als Hauptanwendungsgebiet Data Warehouse-Anwendungen und verteilte Datenbanken sieht, sind Aktualisierungen zu drei Zeitpunkten zum und nach dem Transaktionsende möglich (ON COMMIT, ON DEMAND, periodisch (Abschnitt 4.3.6)). In Abhängigkeit von der Komplexität der Ableitungsvorschrift einer Sicht werden ihre Fakten inkrementell aktualisiert oder rematerialisiert ('full refresh mode'). Für eine inkrementelle Aktualisierung werden nur drei syntaktisch einfach strukturierte materialisierte Sichten unterstützt: Equi-Joins ('materialized join view'), Aggregationen ('materialized aggregate view'), EXISTS-

[4]Diese Version ist auch bekannt als Oracle 8i.

Unteranfragen ('materialized subquery view'). Ferner sind weder UNION-, MINUS- noch OR-Operatoren zugelassen. Aggregatfunktionen und Unteranfragen können nur sehr eingeschränkt geschachtelt werden. Dieser kurze Auszug aus den Einschränkungen[5] zeigt, daß materialisierte Sichten mit inkrementeller Aktualisierung noch nicht für jede Problemstellung im Rahmen von Data Warehouse-Anwendungen und Replikationen eingesetzt werden können. Basis für den inkrementellen Ansatz sind die sogenannten Log-Relationen (Δs, 'log files'). Um die korrekte Aktualisierung zu verschiedenen Zeitpunkten garantieren zu können, wird jede Basisrelation, Sicht und Log-Relation systemintern um ein Zeitattribut (Zeitstempel der Transaktion) ergänzt. Ein Fakt wird aus einer Log-Relation erst gelöscht, wenn auch die letzte abhängige Sicht aktualisiert ist. Die Propagierung der Basisfaktenänderungen einer Transaktion erfolgt reihenfolgeunabhängig als Menge. Da es je Relation nur eine Δ-Relation gibt, verfügt sie über ein Attribut mit einem Kennzeichen für die Änderungsart.

Beispiel 4.2.5 (RA-Δ-Ausdruck bei Oracle)
Ist ein Equi-Join als RA-Ausdruck $s = p \bowtie q$ formuliert, und bezeichnen q^{old}/p^{new} den alten/neuen Zustand der Relationen, dann ist der zugehörige Δ-RA-Ausdruck für die induzierten s-Änderungen ein Ausdruck der Form:

$$\Delta s := \Delta p \bowtie q^{old} \cup p^{new} \bowtie \Delta q.$$

Die in Abschnitt 4.2.1 diskutierte Ineffizienz bei der Anwendung der Δ-Regeln aufgrund mehrfacher Ableitungen der neuen Fakten wird bei Oracle vermieden, da einmal auf der neuen und einmal auf der alten Faktenmenge ausgewertet wird. Da der Ableitungsvorschrift Effektivitätstests fehlen, werden (scheinbar) nur sichere Änderungen abgeleitet. Die physische Eindeutigkeit eines Tupels trotz logischer Duplikate wird durch die systeminterne ROWID gewährleistet, die die physikalische Speicheradresse darstellt, und um die jedes Tupel ergänzt wird. Diese Annahme garantiert einen korrekten Umgang mit logischen Duplikaten ohne die bei [GMS93], [GL95] erforderlichen Verfahrensanpassungen (Abschnitt 4.3.2).

Beim DB-System **DB2** von bf IBM wird die Speicherung abgeleiteter Fakten nicht als Erweiterung des **Sichtkonzepts** (nur virtuelle Sichten) verstanden, vielmehr werden abgeleitete Fakten in Basisrelationen, den sogenannten SUMMARY TABLEs, gespeichert, wobei die Definition der Basisrelation (CREATE [SUMMARY] TABLE) um einen Anfrageausdruck zur Herleitung der Fakten erweitert wird. Ihr Anwendungsschwerpunkt ist der effiziente Zugriff auf abgeleitete Fakten bei sichtenbezogenen Anfragen auf die operative Datenbasis. Die SUMMARY TABLEs unterliegen sehr ähnlichen Restriktionen wie die MATERIALIZED VIEWs bei Oracle. U.a. sind rekursive materialisierte Sichten nicht definierbar, ebenso wenig sind Mengenoperatoren zulässig. Hierarchien materialisierter Sichten sind ebenfalls nicht formulierbar, so daß keine mehrstufige Änderungspropagierung erforderlich wird. Integritätsbedingungen sind, obwohl das Verständnis einer TABLE zugrunde liegt, für SUMMARY TABLEs auch nicht definierbar.

Der primäre Anwendungsschwerpunkt motiviert unter Berücksichtigung des SQL-Transaktionskonzepts einen neuen Zeitpunkt für die Aktualisierung. Soll während einer Transaktion bereits auf den aktuellen Zustand einer materialisierten Sicht zugegriffen

[5] [Ora99] 7-306, Oracle 8i Tuning: 29-9

werden, dann muß unter Beachtung des SQL-Transaktionskonzepts die Aktualisierung unmittelbar nach der Ausführung einer Basisfaktenänderung durchgeführt werden (IMMEDIATE). Dieser IMMEDIATE-Zeitpunkt ist identisch zu dem des SQL-Integritätskonzepts. Im Rahmen der Sichtenaktualisierung bezeichnet DEFERRED jedoch entgegen der Bedeutung im SQL-Kontext bei der DB2 einen beliebigen Zeitpunkt, zu dem der REFRESH TABLE-Befehl ausgeführt wird.

Hinsichtlich der Sichtenaktualisierung basieren die Implementierungen bei Oracle und DB2 auf den gleichen Strategien (Δ-Relationen, spezialisierten Δ-Ableitungsvorschriften, Transitionsregeln etc.), wie sie im Kontext von Datalog und RA publiziert worden sind. Nur unterliegen die Implementierungen einer Reihe von Einschränkungen. Bei den materialisierten Sichten von DB2 sind die besonderen Möglichkeiten und Anforderungen, die sich aufgrund des SQL-Transaktionskonzept ergeben, berücksichtigt worden. Diese Strategie wird auch bei dem in Kapitel 5 entwickelten Verfahren verfolgt, so daß sich analog zu den Integritätsprüfungszeitpunkten die beiden Aktualisierungszeitpunkte IMMEDIATE und DEFERRED ergeben. Ein detaillierter Vergleich zwischen den Implementierungen der beiden DB-Systeme kommerzieller DBS-Hersteller und den im nachfolgenden Kapitel definierten Verfahren wird in Abschnitt 5.7.1 gegeben.

4.3 Erweiterungen

4.3.1 Aggregatfunktionen

Die inkrementelle Aktualisierung von Attributwerten, die auf Aggregatfunktionen wie COUNT, MIN, MAX, SUM, AVG etc. basieren, lassen sich in den Gesamtablauf einer Änderungspropagierung integrieren. Für Regeln mit Aggregatfunktionen müssen jedoch spezielle Relevanzanalysen durchgeführt sowie spezielle Δ-Regeln generiert werden, die den alten Aggregatwert gemäß einer operatorspezifischen Berechnungsvorschrift ohne vollständige Neuberechnung aktualisieren.

Die Zusammenhänge zwischen einer Änderung und der von ihr induzierten Änderung sind bei Aggregatfunktionen wesentlich komplexer als bisher beschrieben. Wesentlich exaktere Relevanzaussagen und Reaktionen auf die Manipulationen können durchgeführt werden, wenn wie bei den Modifikationen die Attributterme und ihr Auftreten in einer Anweisung berücksichtigt werden. Werden Werte von Attributen manipuliert, die in Selektions- und Gruppierungsbedingungen (SQL: WHERE-, GROUP BY-, HAVING-Klauseln) auftreten, so können induzierte Modifikationen einer oder mehrerer Gruppen oder auch Einfügungen oder Löschungen von Gruppen die Folge sein. Treten die manipulierten Attributwerte ausschließlich in SELECT-Klauseln auf, so müssen bestehende Gruppen geändert werden. Welche grundlegenden Überlegungen für eine inkrementelle Propagierung bei Aggregatfunktionen erforderlich sind, kann hier nur beispielhaft skizziert werden. Für COUNT-Funktionen z.B. kann der alte Wert des Sichtentupels für Einfügungen/Löschungen um jeweils eins in- bzw. dekrementiert werden und bei SUM-Funktionen um den Wert des relevanten Attributs des Δ-Fakts. Bei MIN/MAX-Funktionen kann in einem Vortest mit dem alten Wert des manipulierten Fakts vergleichen werden. Nur wenn der neue Wert kleiner/größer ist als der alte muß mit dem eigentlichen Extremwert verglichen werden.

Griffin/Libkin ([GL95]) haben Lösungen für inkrementelle Aggregataktualisierung im Kontext Relationaler Algebra entwickelt und Gupta et.al. ([GMS93]) in SQL wie auch in Datalog. Da diese Problemstellungen von Ceri/Widom ([CW90]) und Gupta et.al. ([GMS93]) bereits im SQL-Kontext diskutiert wurden, werden sie im Rahmen dieser Arbeit nicht weiter vertiefend erörtert.

4.3.2 Duplikate

Gupta et.al. ([GMS93]: Counting) und Griffin/Libkin ([GL95]) berücksichtigen in ihren Verfahren zur inkrementellen Sichtenaktualisierung Duplikate. Sind Duplikate zugelassen, ist es ausreichend, nur ableitbare Änderungen zu propagieren. In einem solchen Fall ist nicht der Test entscheidend, ob alternative Ableitungswege existieren, vielmehr ist die Information relevant, wie viele solcher alternativen Ableitungswege existieren. Bei beiden Verfahren ist die Duplikatproblematik unterschiedlich gelöst.

Gupta et.al. spezifizieren ihren **Counting-Algorithmus** im Datalog-Kontext ausschließlich für materialisierte Sichten, für linearer rekursive Regeln und für Aggregatfunktionen. Sie haben eine deduktive Implementierung basierend auf Datalog-Δ-Regeln gewählt. Ihre Lösung basiert auf der Annahme, daß jedes Fakt unabhängig von der Anzahl der Duplikate in den Basisrelationen und materialisierten Sichten nur einmal gespeichert und in den Sichten nur einmal abgeleitet wird. Um zu erkennen, wie viele Duplikate von einem Tupel existieren, wird jede Relation systemintern um ein Attribut ergänzt, das die Anzahl der alternativen Ableitungen enthält. Auch die Δ-Relationen werden um ein Attribut für die Anzahl der neu hinzu gekommenen/nicht mehr vorhandenen Ableitungswege erweitert. Das Vorzeichen dieses Zählerwerts spezifiziert, ob es sich um eine Einfügung (positiver Wert) oder um eine Löschung (negativer Wert) handelt. Für die inkrementelle Aktualisierung wird ein spezieller Vereinigungsoperator ($\uplus$) verwendet, so daß der neue Zustand einer Relation R wie folgt berechnet werden kann:

$$\text{Counting-Algorithmus:} \quad R^{new} \;:=\; R^{old} \uplus \Delta R.$$

Bei der ersten Materialisierung wird die Anzahl der alternativen Ableitungswege für jedes Fakt ermittelt. Ist ein später einzufügendes Fakt noch nicht vorhanden, so wird es mit '$anzahl = 1$' eingefügt. Bei bereits vorhandenen Fakten wird der Zählerwert nur entsprechend aktualisiert. Physisch gelöscht wird ein Fakt erst, wenn '$anzahl = 0$' ist.

Griffin et.al. präsentieren einen algebraischen Ansatz basierend auf einer **Multimengen-RA** ($\mathcal{BA}$, 'bag semantic') erweitert um spezielle Operatoren für die Differenz ($\dot{-}$) und Vereinigung ($\uplus$), die sich jedoch in ihrer Semantik von denen von Gupta et.al. unterscheiden. Griffin et.al. haben auch eine Lösung für materialisierte Sichten mit Aggregatfunktionen entwickelt, nicht jedoch für Rekursion. Ihre Grundannahme ist, daß jedes Duplikat explizit bei der ersten Materialisierung in der Relation gespeichert wird. Werden n Δ^+-Fakten abgeleitet, so werden n weitere Duplikate der Relation zugefügt, werden n Δ^--Fakten abgeleitet, so werden n Duplikate gelöscht. Die Berechnung des neuen Zustands einer Relation R wird wie folgt ermittelt:

$$\mathcal{BA}\text{-Algorithmus:} \quad R^{new} \;:=\; (\; R^{old} \;\dot{-}\; \Delta^- R \;) \;\uplus\; \Delta^+ R.$$

Mit dem Ziel inkrementeller Sichtenaktualisierung lassen Salem et.al. und Bello et.al. ebenfalls explizit Duplikate im Rahmen der Änderungspropagierung zu. Wie Salem

et.al. Duplikate bei der prototypischen Implementierung in DB2 behandeln, geht leider nicht aus [SBLC00] hervor. Bello et.al. arbeiten in [BDDF+98] systemintern ausschließlich mit eindeutig identifizierbaren Tupeln, indem sie jedes Tupel um ein Attribut erweitern, die ROWID (physikalische Speicheradresse des Tupels). Auf diese Weise ist kein spezieller Algorithmus zur Behandlung von Duplikaten erforderlich.

4.3.3 Modifikationen

Im wesentlichen hat nur die Forschungsgruppe um Olivé, Urpí und Teniente seit 1992 ([UO92], [UO94], [TO95], [MT00]) Lösungen für die Propagierung von Modifikationen entwickelt und die Spezialisierung von Integritätsbedingungen für Modifikationen erörtert. Im allgemeinen werden Modifikationen jedoch als Einfügung eines Fakts mit den neuen Werten und als Löschung des Fakts mit den alten Werten behandelt. Ceri/Widom lassen zwar in [CW90], [CW91] ebenfalls Modifikationen als Basisfaktenänderungen zu, beim ersten Propagierungsschritt werden sie jedoch auch in induzierte Einfügungen und Löschungen aufgespalten.

Die Berücksichtigung von Modifikationen in inkrementellen Verfahren ist ungleich komplexer als bei Einfügungen und Löschungen. Die Vermeidung der Aufteilung in zwei Änderungen stellt jedoch eine weitere Möglichkeit zur Effizienzsteigerung dar. Der entscheidende Vorteil ist eine attributbezogene Relevanzanalyse und somit eine wesentlich exaktere Aussage hinsichtlich der Relevanz von Sichten und Integritätsbedingungen, die zur Laufzeit die Zahl irrelevanter Propagierungen/Prüfungen nochmals beachtlich reduzieren kann. Das generische Verfahren aus Abschnitt 4.1.2 kann ohne wesentliche Veränderungen des Verfahrensablaufs um die Propagierung von Modifikationen erweitert werden. Angepaßt werden müssen jedoch die Transitionsregeln. Zudem müssen zusätzliche $\Delta^{\pm}$-Mengen und sehr spezielle Propagierungsregeln generiert werden.

Zusätzlich zu den Δ^{+} und Δ^{-}-Relationen kann eine $\Delta^{\pm}$-Relation generiert werden mit einer erweiterten Attributstruktur, so daß die neuen und alten Werte eines Fakts gespeichert werden können. Die Relationen $\Delta^{\pm+}$, $\Delta^{\pm-}$ stellen Projektionen auf die neuen/alten Attributwerte von $\Delta^{\pm}$ dar.

Definition 4.3.1 ($\Delta^{\pm}$-Relationen)
Sei $p \in \mathcal{PS}$ das Symbol einer Datalog-Relation mit n Attributen.

- *Die $\Delta^{\pm}$-Relation von p ist eine Relation mit dem Relationssymbol $\Delta^{\pm}p$ und $2 * n$ Attributen. Die ersten n Attribute sind für die neuen Werte des modifizierten Fakts und die letzten n Attribute für die alten Werte.*

- *Der Ausdruck $\Delta^{\pm}p\ (t_1^{new}, ..., t_n^{new},\ t_1^{old}, ..., t_n^{old})$ ist das Literal der $\Delta^{\pm}$-Relation von p, wobei t_i^{new} ein Term für den neuen Wert und t_i^{old} ein Term für den alten Wert des i-ten Attributs des zu modifizierenden Fakts ist $(1 \leq i \leq n)$.*

- $\Delta^{\pm+}p\ (t_1^{new}, ..., t_n^{new}) \leftarrow \Delta^{\pm}p\ (t_1^{new}, ..., t_n^{new},\ t_1^{old}, ..., t_n^{old})$
  $\Delta^{\pm-}p\ (t_1^{old}, ..., t_n^{old}) \leftarrow \Delta^{\pm}p\ (t_1^{new}, ..., t_n^{new},\ t_1^{old}, ..., t_n^{old})$.

Griefahn hat das Konzept der inkrementellen Transitionsregeln für Einfügungen und Löschungen definiert. Sind Modifikationen zugelassen und werden $\Delta^{\pm}$-Relationen verwendet, so ist nur eine Anpassung der Definition für die direkten Transitionsregeln erforderlich, nicht aber für die indirekten.

Definition 4.3.2 (Direkte Transitionsregel für Modifikationen)
Sei $\mathcal{R}$ die stratifizierbare Regelmenge einer Datalog-Datenbank $\mathcal{D}^{\mathcal{D}}$. Die Exponenten ^{old},^{new} bezeichnen den alten/neuen Zustand einer Relation und Δ^* die Δ-Relationen für einzufügende, zu löschende und zu modifizierende Fakten ($* \in \{^+, ^-, ^{\pm+}, ^{\pm-}\}$).
Ist B eine Basisrelation ($B \in \mathcal{B}$), dann ist eine direkte Transitionsregel zur Simulation des neuen Zustands B^{new} ein Ausdruck der Form

$$\begin{aligned} B^{new} &\leftarrow B^{old} \wedge \neg\Delta^- B \wedge \neg\Delta^{\pm-} B \\ B^{new} &\leftarrow \Delta^+ B \\ B^{new} &\leftarrow \Delta^{\pm+} B. \end{aligned}$$

Im weiteren werden nur effektive Modifikationen diskutiert. Die Ergebnisse sind ohne viel Aufwand auf die weniger komplexen Änderungsklassen potentieller, sicherer oder ableitbarer Modifikationen übertragbar. Für die Propagierung effektiver Modifikationen muß bei den Effektivitätstests für die alten Werte des Fakts der gleiche Test wie bei Löschungen durchgeführt werden und für die neuen Werte der gleiche Test wie bei Einfügungen. Der Ableitungsaufwand bei einer Aufspaltung einer Modifikation in eine Einfügung und eine Löschung ist somit identisch zu dem Aufwand der Propagierung einer Modifikation:

$\pm(M^{new}, M^{old})$ ist eine echte induzierte Modifikation gdw.
$\quad (M^{new} \in \mathcal{F}^{new} \wedge M^{new} \notin \mathcal{F}^{old}) \wedge (M^{old} \notin \mathcal{F}^{new} \wedge M^{old} \in \mathcal{F}^{old}).$

Wesentliche Schwierigkeit bei der Propagierung einer Modifikation ist, zu erkennen, wann sie eine Modifikation induziert und wann eine Einfügung bzw. Löschung.

Beispiel 4.3.3 (Propagieren von Modifikationen)
Sei die Regel $s(X,Y) \leftarrow r(X,Y) \wedge t(X,13)$ gegeben und die Basisfakten $r(1,4), r(2,8)$, $t(1,11), t(2,13), t(3,13)$. Das ableitbare Fakt ist $s(2,8)$.

1. $\pm(t(1,13), t(1,11))$ *induziert:* $+s(1,4)$
2. $\pm(t(3,11), t(2,13))$ *induziert:* $-s(2,8)$
3. $\pm(r(3,8), r(2,8))$ *induziert:* $\pm(s(3,8), s(2,8))$
4. $\pm(r(2,7), r(2,8))$ *induziert:* $\pm(s(2,7), s(2,8))$

Die Lösung von Olivé et.al. setzt voraus, daß für jede Relation ein Schlüssel definiert ist. Sind die Werte von Schlüsselattributen der induzierten Modifikation unverändert, so wird eine Modifikation induziert. Werden auch Werte von Schlüsselattributen der Sicht verändert, so werden eine Einfügung und Löschung induziert. Wenn für Beispiel 4.3.3 angenommen wird, daß das erste Attribut der Schlüssel von s ist, dann würden folgende vier induzierten Modifikationen propagiert: $\pm(s(1,4), s(\_,\_))$, $\pm(s(\_,\_), s(2,8))$, $\pm(s(3,8), s(2,8))$, $\pm(s(2,7), s(2,8))$, wobei s(_,_) ein nicht existierendes Fakt bezeichnet. Da sich in den ersten drei Fällen der Wert des Schlüsselattributs verändert, werden sie

als induzierte Einfügungen ($+s(1,4)$, $+s(3,8)$) und Löschungen ($-s(2,8)$) behandelt. Lediglich die induzierte Modifikation im vierten Fall wird nicht aufgeteilt. Aufgrund dieser Zusammenhänge formulieren Olivé et.al. folgende Δ-Regeln für Modifikationen. Wegen einer besseren Übersichtlichkeit verwenden sie dabei für Terme eine abkürzende Vektorenschreibweise (Terme von Schlüsselattributen: $\mathbf{k^{new}}, \mathbf{k^{old}}$, von Nicht-Schlüsselattributen: $\mathbf{t^{new}}, \mathbf{t^{old}}$):

$$\Delta^{\pm} p(\mathbf{k^{new}}, \mathbf{t^{new}}, \mathbf{k^{old}}, \mathbf{t^{old}}) \leftarrow \; p^{new}(\mathbf{k^{new}}, \mathbf{t^{new}}) \wedge p^{old}(\mathbf{k^{old}}, \mathbf{t^{old}}) \wedge \mathbf{k^{new}} = \mathbf{k^{old}}$$

Die Voraussetzung unveränderter Schlüsselattribute garantiert die Effektivität der induzierten Modifikation, so daß auf die ggf. teure Ausführung von Effektivitätstests zur Laufzeit verzichtet werden kann. Die zur Schemaentwurfszeit erforderliche Analyse der Schlüsselattribute ist ggf. mit wenig Aufwand durchführbar. Die Notwendigkeit zur Definition eines Schlüssels stellt jedoch hinsichtlich des Schemaentwurfs eine nicht unbedeutende Restriktion dar, da für alle Basisrelationen und Sichten Schlüssel definiert werden müssen. In vielen kommerziellen DB-Systemen (u.a. Oracle, DB2) sind jedoch Schlüssel für Sichten nicht definierbar. Auch im SQL3-Standard sind die einfach zu analysierenden PRIMARY- und UNIQUE-KEY-Klauseln nicht für Sichten formulierbar. Eindeutigkeitsbedingungen können nur als sehr aufwendig zu analysierende ASSERTIONs definiert werden. Zudem ist das Äquivalenzproblem zweier Formeln unendscheidbar.

Motiviert durch die mangelnde praktische Anwendbarkeit der Lösung von Olivé et.al. im SQL-Kontext werden im Rahmen dieser Arbeit **allgemeinere Vorschriften** für die Propagierung von Modifikationen definiert, die ausschließlich auf den syntaktischen Eigenschaften der Deduktionsregeln basieren und nicht auf den Eigenschaften der Relationen. Die Modifikation einer Relation, deren Literal im Rumpf einer Regel auftritt,

- induziert keine Änderung, wenn Werte von Attributpositionen geändert werden, deren Bereichsvariablen weder in Residuen noch im Kopfliteral auftreten,
- induziert eine Modifikation, wenn sich ausschließlich Werte von Attributpositionen eines Rumpfliterals ändern, deren Variablen nicht in Residuen auftreten, wohl aber im Kopfliteral,
- induziert eine Einfügung und Löschung, wenn Werte von Attributpositionen manipuliert werden, deren Variablen in Residuen auftreten.

Zur Schemaentwurfszeit muß eine detaillierte **attributbezogene Abhängigkeitsanalyse** durchgeführt werden, die ermittelt, welche Variablen eines Rumpfliterals im Kopfliteral und welche in den Residuen auftreten. Die erforderlichen Informationen können in Erweiterung des Konzepts des Abhängigkeitsgraphen für Relationen (Definition 2.1.12) in einem attributbezogenen Abhängigkeitsgraphen (Definition 2.1.13) abgebildet werden. Zur Laufzeit wird dann im Rahmen einer dynamischen Relevanzanalyse (Abschnitt 4.2.5) kontrolliert, welche Attribute einer Relation modifiziert werden und ob die Variable dieser Attributposition im Kopfliteral oder in einer der Residuen auftritt. In Abhängigkeit davon müssen sehr verschiedene Propagierungsregeln ausgeführt werden. Die Hilfsregeln einer optionalen WITH-Klausel werden analog zu dem Vorgehen bei der Formulierung von Δ-Regeln für Einfügungen und Löschungen in Definition 4.1.8 behandelt.

Definition 4.3.4 (Δ-Regeln für Modifikationen)
Sei $R := S \leftarrow L_1 \wedge ... \wedge L_i \wedge ... \wedge L_{n'}$ *WITH* $H \leftarrow L_{n'+1} \wedge ..... \wedge L_n$ *eine Regel einer DB* $\mathcal{D}^{\mathcal{D}}$ *mit einem Rumpfliteral* $rel(L_i) \in rel_{body}(R) \setminus rel_{help}(R)$ $(1 \leq i \leq n)$.
Für L_i *seien die* $\Delta^* L_i$*-Relationen* $(* \in \{^{\pm}, ^{\pm+}, ^{\pm-}\})$ *gegeben. Für* $X \in vars(L_i)$ *seien* X^{new}/X^{old} *die Bereichsvariablen für die entsprechenden Argumente der* $\Delta^* L_i$*-Relationen.*

- *Die Δ-Regel zur Propagierung der durch die Modifikation* $\pm L_i$ *induzierten effektiven Modifikation von* S $(\pm S)$ *ist ein Ausdruck der Form*

$$\begin{aligned}\Delta^{\pm} S \ \leftarrow\ & L_1^{new} \wedge ... \wedge \Delta^{\pm+} L_i \wedge ... \wedge L_{n'}^{new} \wedge\ \neg S^{old} \ \wedge \\ & L_1^{old} \wedge ... \wedge \Delta^{\pm-} L_i \wedge ... \wedge L_{n'}^{old} \wedge\ \neg S^{new} \\ & \textit{WITH}\ \ H^{new} \ \leftarrow\ L_{n'+1}^{new} \wedge ..... \wedge L_n^{new}, \\ & \qquad H^{old} \ \leftarrow\ L_{n'+1}^{old} \wedge ..... \wedge L_n^{old}\end{aligned}$$

gdw. $\forall X \in vars(L_i),\ X^{new}, X^{old} \in vars(\Delta^* L_i) \mid X^{new} \neq X^{old} \Rightarrow$
$X \in vars(S) \ \wedge \ X \notin (\bigcup_{k=1}^{i-1} vars(L_k) \cup \bigcup_{k=i+1}^{n} vars(L_k))$.

- *Die Δ-Regeln zur Propagierung der durch die Modifikation* $\pm L_i$ *induzierten effektiven Einfügung und Löschung von* S $(+S, -S)$ *sind Ausdrücke der Form*

$$\begin{aligned}\Delta^{+} S \ \leftarrow\ & L_1^{new} \wedge ... \wedge \Delta^{\dagger} L_i \wedge ... \wedge L_{n'}^{new} \wedge\ \neg S^{old} \\ & \textit{WITH}\ \ H^{new} \ \leftarrow\ L_{n'+1}^{new} \wedge ..... \wedge L_n^{new}\end{aligned}$$

$$\begin{aligned}\Delta^{-} S \ \leftarrow\ & L_1^{old} \wedge ... \wedge \Delta^{\ddagger} L_i \wedge ... \wedge L_{n'}^{old} \wedge\ \neg S^{new} \\ & \textit{WITH}\ \ H^{old} \ \leftarrow\ L_{n'+1}^{old} \wedge ..... \wedge L_n^{old}\end{aligned}$$

mit $\dagger := {}^{\pm+} \quad \ddagger := {}^{\pm-} \quad$ *falls* $S <_{+1} L_i$
$\dagger := {}^{\pm-} \quad \ddagger := {}^{\pm+} \quad$ *falls* $S <_{-1} L_i$

gdw. $\exists X \in vars(L_i),\ X^{new}, X^{old} \in vars(\Delta^* L_i) \mid X^{new} \neq X^{old}$
$\wedge\ X \in (\bigcup_{k=1}^{i-1} vars(L_k) \cup \bigcup_{k=i+1}^{n} vars(L_k))$.

Die hier entwickelte Strategie basiert auf einer wesentlich umfassenderen syntaktischen Analyse der Strukturen der betroffenen Regel, als dies bei Olivé et.al. gegeben ist. Die Möglichkeiten der Propagierung von Modifikationen, ohne sie in Einfügungen und Löschungen aufzuspalten, sind aufgrund dieser Definitionen wesentlich erweitert. Für noch detailliertere Aussagen bzgl. der Relevanz der Modifikation eines Rumpfliteral muß die aktuell zugrunde liegende Faktenmenge analysiert werden. Die Effizienz solcher Methoden, die sehr viele externe Speicherzugriffe erfordern, ist fraglich und nur empirisch zu bewerten. Die in dieser Arbeit formulierten Vorschriften können um die Vorschriften von Olivé et.al. für die Spezialfälle basierend auf Schlüsseln erweitert werden. Da bei Olivé's Methode auf Effektivitätstests verzichtet werden kann, können somit für Spezialfälle weitere Effizienzsteigerungen bei der Anwendung erreicht werden.

Obwohl der Analyseaufwand zur Schemaentwurfszeit bei der Generierung von Propagierungsregeln wesentlich höher ist als bei Einfügungen und Löschungen und der eigentliche Aufwand für den Propagierungsschritt zur Laufzeit genauso hoch ist wie bei

Einfügungen und Löschungen, so spricht die aufgrund der Attributbezogenheit wesentlich exaktere Relevanzanalyse doch für eine Propagierung von Modifikationen. Dieser Effizienzgewinn gilt nicht nur im Rahmen der Propagierung sondern auch für eine inkrementelle Integritätsprüfung und Sichtenaktualisierung.

Die Definition spezialisierter Integritätsbedingungen für das generische Verfahren (Abschnitt 4.1.2, Definition 4.1.9) muß nur geringfügig erweitert werden, wenn Modifikationen zugelassen sind. So sind Allaussagen der Form $IC \leftarrow \neg H$ nicht nur für Δ^+-Literale spezialisiert sondern auch für $\Delta^{\pm+}$-Literale und werden somit für die neuen Werte modifizierter Fakten angewendet:

$$\Delta IC \leftarrow \neg \Delta^* H, \quad \text{mit } \Delta^* H \text{ mit } * \in \{^+, ^{\pm+}\}.$$

4.3.4 Transitionale Integritätsbedingungen

In der Literatur ([Dec86], [LST86], [KSS87], [CW90], [Kue91], [QW91], [UO92], [TO95], [MT99], [MT00]) werden vielfach nur statische Integritätsbedingungen diskutiert. Aufgrund der für inkrementelle Integritätsprüfungen relevanten Hilfsmittel wie Δ-Mengen und Transitionsregeln bietet sich jedoch eine Erweiterung des Integritätskonzepts hinsichtlich der Formulierung transitionaler Bedingungen an. Bei der Generierung spezialisierter Bedingungen muß lediglich die spezielle Syntax beachtet werden. Der eigentliche Prüfungsprozeß ist unabhängig davon, ob auch transitionale Bedingungen evaluiert werden. Olivé ([Oli91]) und Bry et.al. ([BMM92]) haben inkrementelle Verfahren sowohl für statische als auch für transitionale Integritätsbedingungen entwickelt.

Olivé erweitert die Syntax transitionaler Integritätsbedingungen derart, daß als Rumpfliterale auch die Literale von Δ-Relationen ('internal events': $\Delta^+ L_i, \Delta^- L_i$) zugelassen sind. Auf diese Weise ist die Evaluierung unmittelbar an die Ausführung einer Änderungsanweisung geknüpft. Die transitionalen Integritätsbedingungen sind somit nicht deklarativ formulierbar. Sind auch Modifikationen zugelassen, dann kann ein zu prüfender Zustandsübergang ggf. durch verschiedene Änderungsarten hervorgerufen werden (Einfügung und Modifikation bzw. Löschung und Modifikation). In einem solchen Fall müssen für eine Konsistenzanforderung zwei transitionale Integritätsbedingungen formuliert werden. Für eine Spezialisierung sind jedoch keine weiteren Transformationen erforderlich. Eine Integritätsbedingung ist bei [Oli91] ein Ausdruck der Form:

$$IC \leftarrow L_1 \wedge \ldots \wedge L_n \qquad \text{mit} \quad L_i \in \{L_i,\ \Delta^+ L_i,\ \Delta^- L_i\},\ 1 \leq i \leq n.$$

Die bei Bry et.al. in [BMM92] verwendete Syntax mit Literalen, die die alten (L_i^{old})/ neuen (L_i^{new}) Faktenmengen repräsentieren, erhält den deklarativen Charakter der transitionalen Bedingungen. Liegt der alte Zustand vor, so wird der neue mittels Transitionsregeln simuliert. Übertragen auf die hier verwendete Regelschreibweise ist in [BMM92] eine Integritätsbedingung ein Ausdruck der Form:

$$IC \leftarrow L_1 \wedge \ldots \wedge L_n \qquad \text{mit} \quad L_i \in \{L_i^{new},\ L_i^{old}\},\ 1 \leq i \leq n.$$

Diese Form transitionaler Integritätsbedingungen erfordert, daß für eine inkrementelle Evaluierung die Bedingungen bzgl. der Δ-Relationen spezialisiert werden. Nur die L_i^{new}-Literale sind durch die entsprechenden Δs zu ersetzen. Die L_i^{old}-Literale beziehen sich lediglich auf eine Vergleichsfaktenmenge, die durch entsprechende Variablenbindungen

mit L_i^{new}-Variablen eingeschränkt wird.

In [UO94], [TO95] erweitern Olivé et.al. ihr Integritätskonzept um vier vordefinierte Bedingungen der Inklusionsbeziehung, der Exklusionsbeziehung, der alternativen Schlüssel ('unique keys') und der referenziellen Integrität. Vor einer Spezialisierung werden diese vordefinierten Integritätsbedingungen in semantisch äquivalente regelbasierte Datalog-Integritätsbedingungen transformiert.

4.3.5 Integritätserhaltung

Im Rahmen des Datalog-Transaktionskonzepts ist als Reaktion bei Integritätsfehlern nur das Zurückrollen (ROLLBACK) aller Änderungen der Transaktion vorgesehen. Diese Reaktion ist u.U. ineffizient, da ggf. korrekte Änderungen ebenfalls rückgängig gemacht werden. Eine funktionelle Erweiterung stellt die Korrektur des aufgetretenen Integritätsfehlers dar (Integritätssicherung, 'integrity repair', 'integrity maintenance'). Diese Korrektur ist i.d.R. von der fehlererzeugenden Änderung abhängig. Das deklarative Konzept der Integritätsprüfung wird mit dem prozeduralen Konzept der Fehlerkorrektur kombiniert.

Ceri/Widom haben in [CW90] ein einfaches Verfahren zur Fehlerkorrektur basierend auf Starburst-Triggern entwickelt. Bei der Generierung der Integritätstrigger mit den spezialisierten Formeln der Integritätsbedingungen hat der Entwickler die Möglichkeit, fehlerkorrigierende Änderungsanweisungen bzgl. Basisrelationen im Aktionsteil zu programmieren. Der Analyseaufwand zur Bestimmung der richtigen Korrekturmaßnahme obliegt allein dem Entwickler.

Wesentlich detaillierter werden die Probleme bei der automatischen Generierung von Korrekturanweisungen in [ML91], [Ger96], [MT99], [MT00] erörtert. Moerkotte/Lockemann ([ML91]) und Mayol/Teniente ([MT99], [MT00]) stellen Verfahren vor, die den Entwickler mit systemgenerierten Änderungsanweisungen bei der Fehlerkorrektur unterstützen. Abhängig von einer fehlererzeugenden Änderung werden für eine Integritätsbedingung alternative Korrekturanweisungen ermittelt. Problematisch sind insbesondere Integritätsbedingungen, die für Sichten definiert sind. Da die Abbildung einer Sichtenänderung auf eine bzw. eine Menge von Basisfaktenänderungen nicht immer eindeutig möglich ist, leiten Moerkotte/Lockemann anhand der Sichtdefinitionen Vorschläge für fehlerkorrigierende Basisfaktenänderungen ab. Das Verfahren von Mayol/Teniente basiert auf dem Auftreten von Schlüsseln der Basisrelationen auch in den Sichten, so daß die Zuordnung zwischen Sichten-/Basisfaktenänderung vielfach eindeutig ist. Die Schwierigkeit der Abbildung einer Sichtenänderung auf die zugehörigen Basisfaktenänderungen ist unabhängig von der 'integrity repair'-Thematik als 'view updating'-Problem bekannt.

4.3.6 Zeitpunkte der Sichtenaktualisierung

Für das generische Verfahren wurde nur das Transaktionsende (ON COMMIT) als Zeitpunkt für die Sichtenaktualisierung angenommen. Die Verfahren von Salem et.al. ([SBLC00]) und Bello et.al. ([BDDF+98]) behandeln zwei weitere Zeitpunkte nach dem Ende einer Transaktion: 'periodically' und ON DEMAND (Abschnitt 4.3.7). Die Idee beider Verfahren basiert auf der Annahme, daß materialisierte Sichten nicht Bestandteile der operativen Datenbasis sind, sondern einen zwar abgeleiteten, aber doch separaten Datenbestand

darstellen, wie z.B. bei Data Warehouses. In einem solchen Fall kann ein an die Aktualitätsanforderungen ebenso wie an die Systembelastungen angepaßter Aktualisierungszeitpunkt gewählt werden. Zur Koordinierung unterschiedlich aktueller Faktenmengen einander referenzierender Sichten basieren beide Verfahren auf Zeitstempeln, die für jedes Fakt die zugehörige Transaktionsendezeit spezifizieren. Für jede Sicht wird der Zeitpunkt ihrer Aktualisierung gespeichert. Beide Angaben zusammen erlauben, eine Sicht inkrementell zu einem beliebigen Zeitpunkt nach Beendigung der Transaktion zu aktualisieren. Die Δ-Relationen werden über einen wesentlich längeren Zeitraum als Zwischenpuffer verwendet. Erst wenn die letzte Sicht aktualisiert ist, die eine Relation R (indirekt) referenziert, können die Δ-Fakten von R mit einem Zeitstempel kleiner dem der Sicht, die als erstes aktualisiert wurde, gelöscht werden.

Colby et.al. setzen sich in [CGLMT96], [CKLMR97] ebenfalls mit diesen drei Zeitpunkten auseinander. Zu den 'deferred'-Verfahren zählen sie die periodische und die ON DEMAND-Anpassung. Den Zeitpunkt zum Transaktionsende bezeichnen sie als 'immediate'. Unter Berücksichtigung der Bedeutung dieses Begriffs im SQL-Kontext wird hier weiterhin die Bezeichnung ON COMMIT verwendet. Aus den Definitionen in [CGLMT96] geht eindeutig hervor, daß die Sichten zum Ende einer Transaktion aktualisiert werden. In [CKLMR97] ist nicht erkennbar, ob eine Anpassung unmittelbar im Anschluß an die Ausführung einer Anweisung oder einer Transaktion gemeint ist. Da das Verfahren nur exklusiv zu einem der beiden Zeitpunkte angewendet werden kann, ist von dieser Fragestellung nur der Nettoeffekt der Transaktion betroffen.

Der Ansatz von Colby et.al. unterscheidet sich von dem in Kapitel 5 entwickelten SQL-basiertem Verfahren insbesondere dadurch, daß nicht sowohl während der Transaktion wie auch zu ihrem Ende die Sichten angepaßt werden können. In [CKLMR97] werden zur Verwaltung unterschiedlich aktueller Faktenmengen einander referenzierender Sichten keine Zeitstempel verwendet, sondern der Abhängigkeitsgraph der Sichten wird je nach Aktualisierungszeitpunkt in Untergruppen gegliedert. Die Anforderungen bzgl. des Zeitpunkts, die einander referenzierende Sichtgruppen erfüllen müssen, sind sehr restriktiv bzgl. der zulässigen Struktur eines DB-Schemas, und damit ist die Lösung eher von theoretischem Interesse.

4.3.7 Orthogonale Optimierungen

Die Zeit- und Kostenersparnis der vorgestellten Verfahren resultiert allein aus dem inkrementellen Ansatz. Alle Verfahren können mit den in diesem Abschnitt vorgestellten Konzepten zusätzlich optimiert werden. Dieses Thema der Effizienzoptimierung kann aber nicht ausführlich diskutiert werden, da es über den Rahmen dieser Arbeit hinausgeht. Denkanstöße und Eindrücke von der Komplexität der Optimierung inkrementeller Änderungspropagierung sollen jedoch gegeben werden.

Der allgemeine Vorteil materialisierter Sichten, die Zeitersparnis bei Anfragen durch explizite Speicherung der Fakten, legt nahe, auch für Prozesse zur inkrementellen Integritätsprüfung und Sichtenaktualisierung, Performanz-Vorteile durch die gezielte systeminterne Speicherung von Δ-Fakten zu erzielen. Bei einem 'bottom up' organisierten Prozeß würde die Entscheidung, welche Δ-Mengen zu materialisieren sind, von der Propagierungskomponente getroffen und bei einem 'top down'-Prozeß von der Anfragekomponente

(Abschnitt 4.2.2).

Es ist ebenso schwierig wie aufwendig, die Kriterien für die Entscheidung zu definieren, wann eine Materialisierung mit externen Speicherzugriffen gegenüber wiederholter Auswertungen bei Anfragen sinnvoll ist. Rein syntaktische Kriterien, wie die Anzahl der materialisierten Sichten/Integritätsbedingungen, die eine Sicht referenzieren, oder die Größe der Relationen stoßen sehr schnell an ihre Grenzen. Für weitere Optimierungen sind Kriterien gefordert, die die Semantik betreffen und sich mit der Zeit verändern können. Dazu gehören u.a. die Selektivität der Bedingungen und die Eigenschaften, wie häufig Manipulationen auftreten. Solche Kriterien lassen sich teilweise nur anhand von Statistiken oder Heuristiken qualifizieren.

Ross et.al. ([RSS96]) definieren im SQL-Kontext ein vollständig kostenbasiertes Entscheidungsverfahren auf der Basis der Δ-Relationen und weisen explizit auf die praktische Relevanz ihres Optimierungsverfahrens für eine inkrementelle Integritätsprüfung in SQL-basierten DB-Systeme hin. Kotidis/Roussopoulos stellen in [KR99] ein Verfahren vor, bei dem zur Laufzeit die Entscheidung bzgl. der Materialisierung von Sichten hinsichtlich der Optimierung der Ausführung von Anfragen und inkrementellen Aktualisierungen erfolgt.

Anwendungsspezifisch ist im Rahmen der Sichtenaktualisierung zu entscheiden, welche Kosten (Änderungspropagierung vs. Rematerialisierung) zu welchem Zeitpunkt (während der operativen Arbeit vs. nachts/wochenends) akzeptabel sind, und welcher Grad der Aktualität (Echtzeit vs. historisch) erforderlich ist. Colby et.al. ([CGLMT96], [CKLMR97]), Salem et.al. ([SBLC00]) und Bello et.al. ([BDDF+98]) haben Propagierungsverfahren entwickelt, die es erlauben, daß Sichten zu drei unterschiedlichen Zeitpunkten inkrementell aktualisiert werden können, zum Transaktionsende (ON COMMIT) oder zu einem der beiden Zeitpunkte nach dem Ende der Transaktion (periodisch (Snapshot), auf Anforderung (ON DEMAND)). Die Verfahren sind bereits im Zusammenhang mit dem Zeitpunkt für eine Sichtenanpassung detailliert erläutert worden (Abschnitt 4.3.6). Können auf der Basis solcher Strategien Belastungsspitzen vermieden und eine gleichmäßigere Systembelastung erreicht werden, so kann der Effizienzgewinn aufgrund kürzerer Bearbeitungszeiten für zeitkritische Anwendungen und eingesparter Hardware-Kosten beachtlich sein.

Für eine weitere orthogonale Optimierungsidee, die 'rolling propagation', verwenden Salem et.al. in [SBLC00] Zeitstempel mit der Transaktionsendezeit für jedes Fakt. Der gesamte Prozeß, der alle Änderungen seit der letzten Aktualisierung propagiert, wird systemintern in eine Folge von Teilprozessen aufgespalten, die jeweils nur ein kleines Zeitinterval behandeln. Die Aufspaltung eines großen Intervalls in eine Sequenz kleinerer Intervalle hat zur Folge, daß jeder einzelne Prozeß nur eine Teilmenge der Änderungen propagiert und somit weniger ressourcenintensiv ist und performanter parallel zur operativen Tagesarbeit ausgeführt werden kann.

Ein ebenfalls orthogonales Konzept ist die asynchrone Propagierung ([SBLC00]), bei der der Propagierungszeitpunkt ('propagation phase') unabhängig ist vom Aktualisierungszeitpunkt ('apply phase'). In der Propagierungsphase werden nur die Δ-Fakten abgeleitet, die erst in der Aktualisierungsphase bzgl. der materialisierten Sichten ausgeführt werden. Durch die funktionale Teilung des Gesamtprozesses ist der Zeit- und Ressourcenbedarf je Teilprozeß weniger kritisch. Zudem können die Teilprozesse verteilt

auf verschiedenen Rechnern ausgeführt werden. Die Koordinierung der unterschiedlichen Aktualitätszustände in den drei Faktenmengen, Basisrelationen, Δ-Mengen, materialisierte Sichten, basiert wiederum auf Zeitstempeln. Aufgrund dieser funktionalen Teilung kann ein sowohl an die Ressourcen wie auch an die betrieblichen Abläufe und an den Informationsgehalt angepaßtes Aktualisierungskonzept entwickelt werden.

Kapitel 5

Änderungspropagierung in SQL

In diesem Kapitel werden Verfahren vorgestellt, die SQL-basierte DB-Systeme funktional um leistungsfähige Konzepte der Integritätsprüfung und um materialisierte Sichten erweitern. Durch eine Implementierung dieser Verfahren werden die Integritätskonzepte kommerzieller Hersteller von DB-Systemen um die Möglichkeit erweitert, multirelationale[1] und Sichten referenzierende Integritätsbedingungen definieren zu können. Das Konzept materialisierter Sichten stellt eine Erweiterung des SQL3-Standards dar. Beliebige Hierarchien virtueller und materialisierter (rekursiver) Sichten werden definierbar sein, was bei den DB-Systemen DB2 und Oracle[2] nur eingeschränkt möglich ist.

Daß die hier entwickelten Verfahren echte funktionale Erweiterungen kommerzieller, SQL-basierter DB-Systeme darstellen, setzt voraus, daß gegebene SQL-Konzepte wie das SQL-Transaktions- und Integritätskonzept nicht modifiziert werden. Auch die für die Implementierungsansätze verwendeten SQL-Trigger werden ohne konzeptionelle Veränderungen zugrunde gelegt. Zwar können sehr viele Ideen und Strategien, die in Kapitel 4 im Kontext von Datalog und RA erörtert wurden, nach SQL übertragen werden, doch keines der dort vorgestellten Verfahren genügt den Anforderungen, die aus dem SQL-Transaktions- und Integritätskonzept resultieren. Lediglich bei der DB2 werden zwei Aktualisierungszeitpunkte (IMMEDIATE und DEFERRED) berücksichtigt, doch dort sind materialisierte Sichten und Integritätsbedingungen in ihrer Funktionalität und Ausdruckskraft gegenüber den hier entwickelten Konzepten eingeschränkt. Daß aus Gründen einer vereinfachten Darstellung das normalisierte NSQL als Anfragesprache verwendet wird, beeinflußt nicht die Anwendbarkeit der Lösungen im SQL3-Kontext. Eine Erweiterung der Verfahren für Sichten mit Aggregatfunktionen und Duplikaten kann gemäß den Verfahren aus den Abschnitten 4.3.1 und 4.3.2 ohne Einschränkungen durchgeführt werden.

Basis der hier entwickelten Verfahren ist die Methode zur inkrementellen Änderungspropagierung von Griefahn in [Grie97], so wie sie bereits beim generischen Verfahren in Abschnitt 4.1.2 verwendet wurde. Die Definitionen spezialisierter Propagierungsvorschriften und Integritätsbedingungen im SQL-Kontext orientieren sich an dem Vorgehen von Griefahn bei der Modellierung von Δ-Regeln. Wie Datalog-Regeln und -Integritätsbedingungen auf NSQL-Sichten und -ASSERTIONs abgebildet werden können, zeigen die Trans-

[1] Multirelationale Integritätsbedingungen sind in kommerziellen DB-Systemen bislang nur als FOREIGN KEY-Beziehungen formulierbar (Abschnitt 2.2.3).

[2] Die Restriktionen der Implementierungen materialisierter Sichten bei Oracle und DB2 werden ausführlich in Abschnitt 4.2.6 erörtert.

formationsvorschriften aus Abschnitt 3.2.1. Spezielle Techniken wie die 'Magic Sets'-Transformation, die Griefahn für eine effiziente Auswertung der Effektivitätstests anwendet, bleiben in dieser Arbeit unberücksichtigt, da dies als Aufgabe der systeminternen Anfragekomponente betrachtet wird.

Der SQL-basierte Ansatz von Ceri/Widom ([CW90], [CW91]) wird nicht als Grundlage für die Definition spezialisierter Ausdrücke verwendet, weil sie bei ihren Definitionen unmittelbar die Besonderheiten einer Implementierung mittels Starburst-Triggern berücksichtigen. Zudem haben sie anders als bei NSQL den zugrunde gelegten SQL-Dialekt nicht normalisiert, so daß die von ihnen durchgeführten Relevanzanalysen und Transformationen nicht unmittelbar mit den aus dem Datalog-Kontext bekannten Vorgehensweisen vergleichbar sind. Eine Anwendbarkeit der aus Kapitel 4 bekannten Verfahren könnte nicht so transparent dargestellt werden. Die von Griefahn im Datalog-Kontext definierten Vorschriften für Einfügungen und Löschungen werden um die Möglichkeit der Propagierung von Modifikationen erweitert. Dabei werden die grundlegenden Erkenntnisse der Methoden von Urpí/Olivé ([UO92]) und Mayol/Teniente ([MT95]) zwar übernommen, jedoch ungeachtet des Spezialfalls vorhandener Schlüsselattribute formuliert.

Um die Entscheidung für einen triggerbasierten Realisierungsansatz zu motivieren, werden in Abschnitt 5.1 alternative Lösungen skizziert und bewertet. Das in Abschnitt 5.2 definierte Propagierungsverfahren stellt die Basis dar für die in den nachfolgenden Abschnitten durchgeführten funktionalen Erweiterungen um Integritätsprüfung (Abschnitt 5.3), Sichtenaktualisierung (Abschnitt 5.4) und um die simultane Ausführung beider Aufgaben (Abschnitt 5.5). Die für die Durchführung der Verfahren erforderlichen DB-Objekte werden, wie auch die Abläufe während einer Transaktion, anhand eines detaillierten Beispiels in Abschnitt 5.6 verdeutlicht. Abschließend werden die zuvor entwickelten Konzepte und Implementierungsansätze in Abschnitt 5.7 mit anderen Verfahren verglichen.

Die weiteren Diskussionen werden im SQL-Kontext geführt. Statt den Präfixen 'SQL3' und 'NSQL' wird der allgemeinere Präfix 'SQL' verwendet bzw. ganz auf ihn verzichtet, da in den hier diskutierten Punkten SQL3 und NSQL übereinstimmen und die Aussagen somit für beide DB-Sprachen gleichermaßen gelten. Wenn eine Differenzierung zwischen SQL3 und NSQL erforderlich wird, so wird sie durch den entsprechenden Präfix angezeigt. In Analogie zu der in Kapitel 4 verwendeten Kurzschreibweise für Änderungsanweisungen basierend auf dynamischen Literalen $*M$ mit $* \in \{+, -, \pm\}$ werden in diesem Kapitel die SQL-Änderungsanweisungen INSERT, DELETE, UPDATE ebenfalls durch $*M$ dargestellt, wobei $\{+, -, \pm\}$ die Änderungsart symbolisieren. Es sei darauf hingewiesen, daß Implementierungsansätze erörtert werden. Eine Implementierung ist noch nicht erfolgt. Derzeit wird jedoch das in Abschnitt 5.5 vorgestellte Verfahren im Rahmen zweier Diplomarbeiten auf einem DB2-DB-System realisiert.

5.1 Alternative Implementierungsstrategien

Die Vor- und Nachteile einer Realisierung der Benutzerschnittstelle als Interpretierer oder als Präprozessor werden in Abschnitt 5.1.1 erörtert. Um zu zeigen, daß eine funktionale Erweiterung von SQL-basierten DB-Systemen auf der Basis ihrer bestehenden Funktionalität (Basisrelationen, virtuelle Sichten, Trigger etc.) möglich ist, bieten sich verschiedene

Implementierungsalternativen an: deduktiv mit prozeduraler Ausführungssteuerung oder triggerbasiert. Nach der Diskussion der Vor- und Nachteile deduktiver und triggerbasierter Implementierungen in Abschnitt 5.1.2 werden in Abschnitt 5.1.3 Lösungen für Teilaufgaben der Verfahren basierend auf verschiedenen SQL-Triggertypen (BEFORE/AFTER und ROW/STATEMENT) skizziert und bewertet. Für die Lösungsskizzen und Diskussionen im Rahmen dieses Abschnitts wird das generische Verfahren aus Abschnitt 4.1.2 zugrunde gelegt. Weitere Detailkenntnisse von SQL-Integritäts- und Sichtkonzepten sind nicht erforderlich und werden erst in den nachfolgenden Abschnitten ausführlich erörtert.

5.1.1 Interpretierer vs. Präprozessor

Für die Realisierung der Benutzerschnittstelle bieten sich die Möglichkeiten einer Interpretierer- oder Präprozessor-Lösung an. Der Vorteil eines Interpretierers ist die einheitliche Benutzeroberfläche, so daß die Systemerweiterung für Anwender transparent ist. Vorteil eines Präprozessors ist seine vollständige Integration in das DB-System. Die gewünschte Funktionalität steht beim Präprozessor nicht nur durch die Interpretierer-Schnittstelle zur Verfügung, sondern kann von beliebigen Schnittstellen und Anwenderprogrammen genutzt werden. Daß Anwender die Präprozessor-Prozeduren explizit aufrufen müssen, birgt nur ein geringfügiges Fehlerpotential, da es sich bei der Zielgruppe primär um DB-Entwickler und -Administratoren handelt, und somit ein entsprechendes Verständnis vorausgesetzt werden kann. Da es eine Intention dieser Arbeit ist, zu zeigen, daß die Konzepte des SQL3-Standards leistungsfähig genug sind, um SQL3 funktional um zusätzliche DB-Konzepte zu erweitern, ist für den Implementierungsansatz eine Präprozessor-Lösung gewählt worden. Das Prinzip einer Präprozessor-Lösung wird in der Abbildung 5.1 veranschaulicht. Der Präprozessor sollte trotz seiner Vorteile nur eine Lösung zur funktionalen Erweiterung bereits existierender DB-Systeme sein.

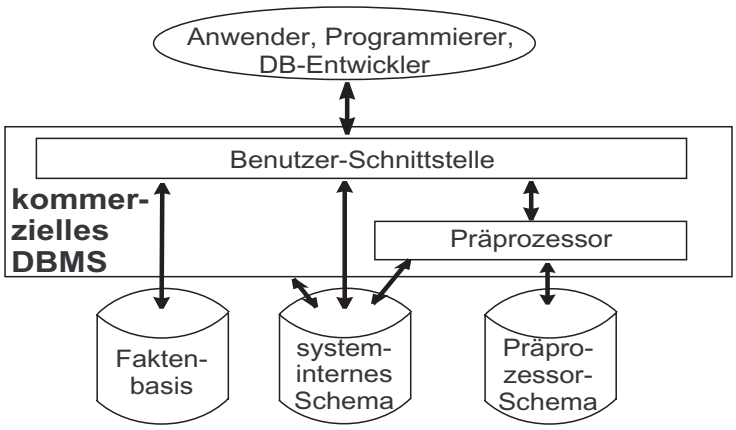

Abbildung 5.1: Präprozessor-Prinzip

Die Möglichkeiten, kommerzielle DB-Systeme funktional zu erweitern, sind sehr beschränkt. Die Programme des DB-Systems können nur von den Herstellern selbst modifiziert werden. Für Anwender bleiben nur die Möglichkeiten zur Erweiterung auf der Basis der systeminternen Schnittstellen wie z.B. den sogenannten Cartridges bei Oracle oder

mittels DB-Prozeduren, -Funktionen und Triggern. Bei beiden Lösungen ist die funktionale Erweiterung für Anwender gleichermaßen transparent. Die Programmierung von Schnittstellenerweiterungen ist jedoch so herstellerspezifisch, daß eine Portabilität nicht mehr gegeben ist. Ein Vorteil einer Realisierung auf der Basis von DB-Prozeduren ist, daß die Präprozessor-Prozeduren je nach verwendeter Programmiersprache auch auf anderen DB-Systemen installiert werden können.

Aus diesen Gründen ist eine Präprozessor-Lösung, die auf der Basis von DB-Prozeduren und -Funktionen definiert wird, gewählt worden. Da weder ASSERTIONs noch materialisierte Sichten in DB-Systemen kommerzieller Hersteller zulässige DB-Objekte sind, stehen keine DDL-Anweisungen zur Verfügung, mittels der sie erzeugt werden können. Ferner sind die für ASSERTIONs zulässigen Formeln keine in einem DB-System zulässigen Bedingungsausdrücke. Im Rahmen des Schemaentwurfs bzw. -evolution ruft der Anwender zur Erzeugung und Modifikation von ASSERTIONs und materialisierten Sichten die erforderlichen Präprozessor-Prozeduren auf und übergibt die Definition als Argument (z.B. *create_assertion*(), *create_materialized_view*()). Um diese neuen DB-Objekte verwalten zu können, muß der Präprozessor neben dem DB-Schema auch auf eigene Schemarelationen zugreifen können, dem Präprozessor-Schema (externes Schema). Die Interaktion zwischen DB-System, Benutzerschnittstelle und Präprozessor sowie ihre Zugriffe auf das DB- und das Präprozessor-Schema werden in Abbildung 5.1 skizziert.

Die zweite Aufgabe des Präprozessors ist die Transformation der DB-Objekte des Präprozessor-Schemas in DB-Objekte des DB-Schemas. Für die ASSERTIONs und materialisierten Sichten werden DB-Objekte des DB-Systems generiert, wie z.B. Basisrelationen, virtuelle Sichten und Trigger, mittels denen die Objekte mit der neuen Funktionalität simuliert werden. Dazu gehört auch die Generierung aller für die Durchführung einer effizienten Integritätsprüfung und Sichtenaktualisierung erforderlichen DB-Objekte. Zu diesem Zweck werden die Abhängigkeiten zwischen allen Relationen und Integritätsbedingungen des internen Schemas analysiert, ein Abhängigkeitsgraph erstellt und im externen Schema abgelegt. Als Ergebnis dieser Analyse werden Relevanzaussagen getroffen, die eine wichtige Information für die Generierung der erforderlichen DB-Objekte darstellen.

Die Ausführung der Präprozessor-Prozeduren während des Schemaentwurfs kann für Anwender nicht vollständig transparent realisiert werden. Da SQL-DDL-Anweisungen zur Erzeugung von ASSERTIONs und materialisierten Sichten in kommerziellen DB-Systemen bzw. in SQL3 fehlen, muß ein Anwender die Präprozessor-Prozeduren zur Pflege solcher DB-Objekte explizit aufrufen. Ruft ein Anwender eine SQL-DDL-Anweisung zur Pflege eines systeminternen DB-Objekts auf, so muß er zusätzlich zum Ende einer Schemaentwurfs- oder -modifikationsphase eine Präprozessor-Prozedur aufrufen, die die für das Verfahren generierten DB-Objekte anpaßt bzw. erzeugt. Da SQL-Trigger nicht für die Ausführung von DDL-Anweisungen feuern, können die Präprozessor-Prozeduren nicht automatisch gestartet werden. Die Anforderungen an eine Schemaevolution sind jedoch nicht Thema dieser Arbeit und werden somit nicht ausführlich diskutiert.

5.1.2 Deduktive vs. aktive Ansätze

Die verschiedenen Auswertungsstrategien 'top down' und 'bottom up' sowie deduktiv und aktiv wurden bereits in Abschnitt 4.2.2 in Hinblick darauf skizziert, wie sie im Kontext von Datalog und RA bzw. mittels Starburst- und ActLog-Triggern realisiert werden können. Im weiteren werden sie hinsichtlich ihrer Anwendung in SQL bewertet.

Deduktiver Implementierungsansatz:

Zentrale Idee eines deduktiven Ansatzes ist die Realisierung spezialisierter Ableitungsvorschriften und Integritätsbedingungen als Δ-Sichten und die Simulation von Zuständen mittels Transitionssichten (Bry et.al. [BMM92], [BDM88], Gupta et.al. [GMS93], Manthey [Man94]). Da in Abschnitt 3.2.1 gezeigt wurde, wie Datalog-Regeln nach SQL transformiert werden können, wird für die Skizze einer SQL-basierten deduktiven Lösung das generische Verfahren aus Abschnitt 4.1.2 zugrunde gelegt.

Beispiel 5.1.1 (Δ-Regel und Δ-Sicht)
Sei die aus dem Beispiel der Unternehmensorganisation (Beispiele 2.1.17, 3.1.4) bekannte Sicht 'mita_abt' gegeben. Für die Propagierung ableitbarer 'mita_abt'-Fakten kann für Einfügungen in die 'mitarbeiter'-Relation gemäß dem Vorgehen beim generischen Verfahren folgende Δ-Regel modelliert werden, die ihrerseits in die nachfolgende SQL-Sicht transformiert werden kann.

Datalog: $\Delta^+ s(X_1, Y_2) \leftarrow \Delta^+ p(X_1, X_2) \wedge q(X_2, Y_2);$

SQL:
```
VIEW Δ+s(a1, a2) AS
     SELECT tvp.a1, tvq.a2 FROM Δ+p AS tvp, q AS tvq
     WHERE tvp.a2 = tvq.a1.
```

Ein **deduktiver 'bottom up'-Ansatz** erfordert neben den Δ-Sichten eine aktive Ausführungssteuerung, die 'änderungsgetrieben' eine stratumbezogene Propagierung realisiert. Zu diesem Zweck kann der für das generische Verfahren vorgestellte Algorithmus (Algorithmus 4.1.2) unmittelbar in SQL angewendet werden. Die Δ-Relationen und materialisierte Sichten können in SQL als Basisrelationen realisiert werden, die Δ-Regeln als Δ-Sichten, die spezialisierten Integritätsbedingungen als spezialisierte Integritätssichten und die Transitionsregeln als Transitionssichten. Für eine Steuerung der Ausführungsreihenfolge können alle (generierten) Sichten und ASSERTIONs gemäß ihren Abhängigkeiten Stratifikationsebenen zugeordnet werden.

Die Steuerungsprozedur kann z.B. als DB-Prozedur implementiert werden. Zu Ihren Aufgaben gehört die Steuerung der stratumbezogenen Ausführungsreihenfolge nur der relevanten Δ-Sichten (Abbildung 5.2). Anhand der Informationen eines Abhängigkeitsgraphen kann entschieden werden, welche Sichten und ASSERTIONs für die gegebenen Änderungen relevant sind. Zudem müssen die Ergebnisse der Anwendung der Δ- und Integritätssichten verarbeitet bzw. interpretiert werden. Die Δ-Fakten sind ggf. zu speichern und die entsprechenden Änderungen auf den Basisrelationen und materialisierten Sichten auszuführen. Um entscheiden zu können, ob ein Propagierungsprozeß beendet werden kann, können zum Ende der Bearbeitung eines Stratums die Inhalte der relevanten Δ-Relationen von der Steuerungsprozedur kontrolliert werden. Wurden keine

neuen Δ-Fakten abgeleitet, kann der Prozeß beendet werden. Die Ausführung eines lokalen ROLLBACKs, wie es bei einer Konsistenzverletzung im Rahmen einer IMMEDIATE-Integritätsprüfung erforderlich ist, ist aufwendig zu simulieren. So muß z.B. vor der Ausführung einer Basisfaktenänderung durch einen Benutzer ein Sicherungspunkt gesetzt werden, zu dem dann im Falle eines IMMEDIATE-Fehlers zurückgerollt werden kann.

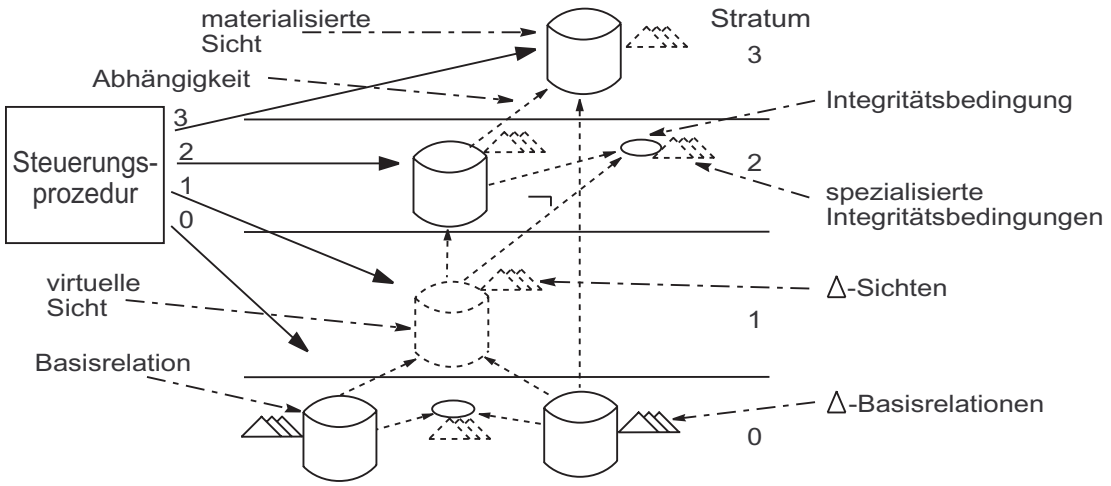

Abbildung 5.2: Skizze einer deduktiven Implementierung

Die Trennung zwischen Ausführungssteuerung (Prozedur) und Propagierungsvorschrift (Δ-Sichtdefinition) ermöglicht eine einfache Schemaevolution. Werden z.B. neue Sichten definiert, so müssen nur neue Δ-Sichten generiert und einem Stratum zugeordnet werden, die Ausführungssteuerung hingegen bleibt unberührt. Problematisch ist aber, daß zum Zeitpunkt der Programmierung der Prozedur nicht bekannt ist, welche Sichten später anzuwenden sind. In SQL3 ist eine 'dynamic SQL'-Schnittstelle[3] für Programmiersprachen definiert worden, die es erlaubt, den auszuführenden SQL-Befehl erst zur Laufzeit in den Programmcode einzubinden. Da die SQL-Anweisungen aber erst zur Laufzeit kompiliert werden, sind die Ausführungen solcher 'dynamischer' Programme wenig performant.

Bei einer **deduktiven 'top down'-Lösung** werden die Δ-Sichtdefinitionen und spezialisierten Integritätsbedingungen anfragegetrieben angewendet und auf die Ergebnisse wird entsprechend reagiert. Von Vorteil ist dabei, daß keine Ausführungsteuerung realisiert werden muß. So können z.B. alle spezialisierten Integritätsbedingungen konjunktiv zu einer Integritätsbedingung verknüpft werden, so daß bei der Integritätsprüfung nur diese eine Bedingung von der Anfragekomponente ausgewertet werden muß. Bei einer solchen Lösung wird das Laufzeitverhalten in besonderem Umfang von den Optimierungen bestimmt, die die Anfragekomponente durchführen kann. Ein solcher Optimierungsschritt wäre z. B. das Erkennen sich wiederholender Teilanfragen und die Wiederverwendung von Zwischenergebnissen der Anfrageauswertung. Dazu kann auch das frühzeitige Erkennen leerer Relationen und ggf. der Abbruch weiter Auswertungen gehören. Für ein umfangreiches DB-Schema mit einer Vielzahl von Sichten und Integritätsbedingungen und einer

[3] [DD97] 20; [GP99] 39; 'Dynamic SQL' ist eine spezielle Funktion der 'Embedded SQL'-Schnittstelle. Die DB-Systeme Oracle und DB2 bieten eine entsprechende Funktionalität an.

umfangreichen Faktenbasis, ist es fraglich, ob die Anfragekomponenten kommerzieller DB-Systeme diesen Anforderungen gerecht werden können. Diese Optimierungsschritte können bei einer 'bottom up'-Propagierung durch eine Relevanzanalyse und eine stratumbezogene Ausführung realisiert werden. Solche 'expliziten' Optimierungen sind für einen 'top down'-Ansatz aufwendiger als bei einem 'bottom up'-Ansatz zu realisieren.

Aktiver Implementierungsansatz:

Triggerbasierte Implementierungen sind bei (Starburst: [CW90], [CW91], [CW94]; ActLog: [Grie97]; Decker: [Dec01], Ceri/Fraternali [CF97]) zu finden. Für eine triggerbasierte Implementierung ist eine auf drei Triggerarten basierende Lösung möglich. Aufgabe sogenannter Propagierungstrigger (PR-Trigger) kann dabei die Ableitung induzierter Änderungen und die Steuerung des änderungsgetriebenen Fixpunktprozesses sein. Eine Δ-Regel, wie sie im Rahmen des generischen Verfahrens formuliert wurde, kann derart in einen PR-Trigger transformiert werden, daß der Regelrumpf der Δ-Regel als Anfrage an die Faktenmenge im Aktionsteil angewendet wird, und die abgeleiteten Δ-Fakten dann in eine Δ-Basisrelation eingefügt werden. Durch die Einfügung in die Δ-Basisrelation werden die PR-Trigger nachfolgender Strata gefeuert und somit die 'bottom up'-Propagierung gesteuert. Der Propagierungsprozeß terminiert automatisch, wenn keine neuen PR-Trigger gefeuert werden, weil z.B. keine neuen Δ-Fakten mehr abgeleitet werden. Für die Integritätsprüfung kann im Aktionsteil sogenannter Integritätsprüfungstrigger (IP-Trigger) die spezialisierte Formel einer ASSERTION angewendet werden, und auf einen Integritätsfehler wird mit einem ROLLBACK reagiert. Die Aktualisierung materialisierter Sichten kann durch sogenannter Sichtenaktualisierungstrigger (SA-Trigger) durchgeführt werden. Gemäß der Änderungsart eines Δ-Fakts führen sie die entsprechende Änderung auf der Faktenmenge der materialisierten Sicht aus. Welche PR-, IP-, SA-Trigger und welche Δ-Basisrelationen bei diesem aktiven Implementierungsansatz vom Präprozessor für ein gegebenes DB-Schema zu generieren sind, zeigt die Abbildung 5.3. Zudem wird die stratumbezogene Ausführung der Trigger dargestellt.

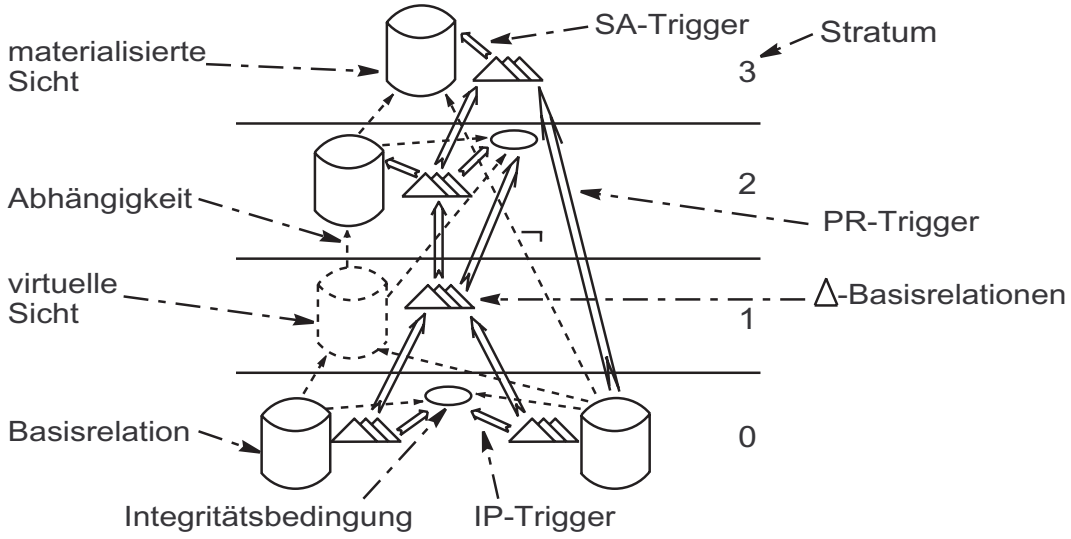

Abbildung 5.3: Skizze einer aktiven Implementierung

Beispiel 5.1.2 (PR-, IP-, SA-Trigger)
Für die Δ-Sicht aus Beispiel 5.1.1 kann folgender PR-Trigger pr_1 generiert werden. Die spezialisierte Formel der Δs-Sichtdefinition zur Ermittlung der $\Delta^+ s$-Fakten tritt als Formel der bedingten Einfügeanweisung auf.

$$\begin{array}{l} \textit{TRIGGER}\ \ pr_1\ \ \textit{AFTER INSERT ON}\ \ \Delta^+ p\ \ \textit{FOR EACH STATEMENT} \\ \quad \textit{BEGIN ATOMIC}\ \ \textit{INSERT INTO}\ \ \Delta^+ s\ \ (a_1, a_2) \\ \qquad\qquad \textit{SELECT}\ \ tv_p.a_1, tv_q.a_2\ \ \textit{FROM}\ \ \Delta^+ p\ \textit{AS}\ \ tv_p,\ \ q\ \textit{AS}\ \ tv_q \\ \qquad\qquad \textit{WHERE}\ \ tv_p.a_2 = tv_q.a_1\ \ \ \textit{END.} \end{array}$$

Wird vorausgesetzt, daß die Δ-Fakten in den Δ-Relationen gespeichert vorliegen, kann zur Einfügung in die $\Delta^+ s$-Basisrelation z.B. folgender SA-Trigger sa_1 generiert werden, sofern s eine materialisierte Sicht ist. Für jedes neue $\Delta^+ s$-Fakt wird ein Fakt mit den Attributwerten von $\Delta^+ s$-Fakten in s eingefügt.

$$\begin{array}{l} \textit{TRIGGER}\ \ sa_1\ \ \textit{AFTER INSERT ON}\ \ \Delta^+ s\ \ \textit{FOR EACH STATEMENT} \\ \quad \textit{BEGIN ATOMIC}\ \ \textit{INSERT INTO}\ \ s\ \ (a_1, a_2) \\ \qquad\qquad \textit{SELECT}\ \ tv_s.a_1, tv_s.a_2\ \ \textit{FROM}\ \ \Delta^+ s\ \textit{AS}\ \ tv_s\ \ \ \ \textit{END.} \end{array}$$

Für eine SQL-ASSERTION ic_1 der Form

$$\begin{array}{l} ic_1\ \textit{CHECK}\ (\ \textit{NOT EXISTS}\ (\ \textit{SELECT}\ tv_p.a_1, tv_p.a_2\ \textit{FROM}\ \ p\ \textit{AS}\ \ tv_p \\ \qquad\qquad\qquad\qquad \textit{WHERE}\ \textit{NOT}\ (\ tv_p.a_2 = tv_p.a_1\ )\ )\ ) \end{array}$$

kann ein IP-Trigger generiert werden, in dessen Aktionsteil die für $\Delta^+ p$ spezialisierte Formel der ASSERTION angewendet wird. Um das Ergebnis der Anwendung interpretieren zu können, tritt die spezialisierte Formel als Bedingung einer IF-Anweisung auf. Als Reaktion wird im Fehlerfall ein lokales ROLLBACK ausgeführt.

$$\begin{array}{l} \textit{TRIGGER}\ \ ip_1\ \ \textit{AFTER INSERT ON}\ \ \Delta^+ p\ \ \textit{FOR EACH STATEMENT} \\ \quad \textit{BEGIN ATOMIC IF}\ (\textit{NOT EXISTS}\ (\ \textit{SELECT}\ tv_p.a_1, tv_p.a_2\ \textit{FROM}\ \ \Delta^+ p\ \textit{AS}\ \ tv_p \\ \qquad\qquad\qquad\qquad\qquad \textit{WHERE}\ \textit{NOT}\ (\ tv_p.a_2 = tv_p.a_1\ )\ )\ ) \\ \qquad\qquad \textit{THEN}\ \ null;\ \ \textit{ELSE}\ \ lokales\_ROLLBACK;\ \ \textit{END.} \end{array}$$

Diese Lösungsskizze stellt nur einen der möglichen triggerbasierten Implementierungsansätze dar, und das Beispiel soll nur einen Eindruck von der Idee der PR-, IP-, SA-Trigger vermitteln. Ihre Struktur wird in den nachfolgenden Abschnitten noch kritisch diskutiert. Diese Idee der Verwendung von PR-, IP-, SA-Triggern wird jedoch allen nachfolgenden Implementierungsansätzen zugrunde gelegt.

Die Vorteile einer triggerbasierten Implementierung basieren darauf, daß von der Triggerkomponente sehr viel Funktionalität bereitgestellt wird, die im 'bottom up'-Propagierungsprozeß benötigt wird. Da Trigger unmittelbar für die Ausführung einer Änderungsanweisung ausgeführt werden, kann ein IMMEDIATE-Verarbeitungsprozeß automatisch gestartet werden. Das bei einem IMMEDIATE-Integritätsfehler auszuführende lokale ROLLBACK kann sehr einfach durch einen Abbruch der Triggerausführung simuliert werden, da in diesem Fall die Basisfaktenänderung, die den Trigger aktiviert hat, zusammen mit allen durch sie induzierten Änderungen zurückgerollt wird. Die zur Laufzeit auszuführende Relevanzanalyse, die bei der prozeduralen Steuerung beschrieben wurde, kann

vollständig durch die ereignisorientierte Aktivierung der Trigger realisiert werden. Die Anwendung ausschließlich relevanter Propagierungsvorschriften und Integritätsbedingungen wird dadurch gewährleistet, daß für eine Änderung nur relevante Trigger feuern. Der Prozeß terminiert automatisch, wenn keine weiteren Trigger gefeuert werden können.

Bereits diese kurze Verfahrensskizze verdeutlicht einige Nachteile einer triggerbasierten Implementierung. Zum Transaktionsende wird kein SQL-Trigger gefeuert oder ausgeführt, so daß ein automatischer Start der DEFERRED-Verarbeitung nicht möglich ist. Da Trigger nicht für Sichten definiert werden können, müssen alle Δ-Mengen als Basisrelationen realisiert werden. Die Möglichkeit der Performanzoptimierung aufgrund materialisierter oder virtueller Δ-Relationen besteht somit nicht. Das auf der Triggererzeugungszeit basierende Prioritätenkonzept erfordert aufwendige Reaktionen im Falle einer Schemaevolution. Bei jeder Manipulation einer Propagierungsvorschrift oder Integritätsbedingung müssen nicht nur die betroffenen SA- und IP-Trigger neu generiert werden sondern auch alle Trigger, die aufgrund ihrer Erzeugungszeit hinter den neu generierten Triggern auszuführen sind, wozu auch die benutzerdefinierten Trigger zählen.

Aktiver und deduktiver Implementierungsansatz:

Die Struktur des PR-Triggers pr_1 aus Beispiel 5.1.2 legt eine Lösungsalternative basierend auf PR-Triggern und Δ-Sichten nahe. Da die Formel der bedingten Einfügeanweisung die Formel der spezialisierten Ableitungsvorschrift der Δ^+s-Sicht ist (Beispiel 5.1.1), kann eine Lösung basierend auf einer Δ^+s-Sicht und einem modifizierten pr_1-Trigger entwickelt werden. Ist eine Δ^+s-Sicht gegeben, dann kann die INSERT-Anweisung von pr_1 wie folgt verkürzt formuliert werden:

$$\text{INSERT INTO } \Delta^+s \ (a_1, a_2) \text{ SELECT } tv_s.a_1, tv_s.a_2 \text{ FROM } \Delta^+s \text{ AS } tv_s.$$

Eine solche kombinierte Lösung basierend auf Triggern und Sichten (aktiv und deduktiv) hat gegenüber einer ausschließlich triggerbasierten Lösung die Vorteile einer sehr übersichtlichen Trigger-Definition und einer einfachen Schemaevolution. So ist es bei vielen Änderungen der Sichtdefinitionen ausreichend, nur die Definitionen der Δ-Sichten zu manipulieren, ohne die Trigger neu generieren zu müssen.

Da selektierend auf die Δ-Relationen zugegriffen wird, bietet sich die Verwendung von STATEMENT-Triggern an, wobei für jede Einfügung in eine Δ-Relation ein STATEMENT-Trigger mengenorientiert ausgeführt wird. Dieser selektiert die zu verarbeitende Faktenmenge aus der Δ-Relation. Die Nachteile einer kombinierten Lösung entsprechen im wesentlichen den Nachteilen der Verwendung von STATEMENT-Triggern gegenüber ROW-Triggern, wie sie im nachfolgenden Abschnitt 5.1.3 diskutiert werden.

Nur wesentliche Vor- und Nachteile verschiedener deduktiver und aktiver Implementierungsansätze sind aufgeführt worden. Insbesondere für Trigger werden noch weitere Schwierigkeiten und die daraus resultierenden Restriktionen detailliert erörtert. Wesentliches Kriterium der Entscheidung gegen eine deduktive Lösung mit prozeduraler Steuerung ist die Notwendigkeit, dynamisches SQL verwenden zu müssen, wodurch sich die Ausführungszeit während der Transaktion verlängern kann. Zentrales Kriterium für eine triggerbasierte Realisierung ist die weitgehende Unterstützung der Ausführung eines iterativen Fixpunktprozesses durch das Ausführungsmodell der SQL-Trigger. Bei einer

prozeduralen Steuerung müßte sehr viel der vorhandenen Funktionalität erneut programmiert werden. Da aber der Aufwand bei der Schemaevolution die Laufzeit während der Transaktion nicht negativ beeinflußt, wird ein triggerbasierter Ansatz entwickelt. Um die Leistungsfähigkeit und Grenzen der Triggermodelle aufzeigen zu können, werden im Rahmen der triggerbasierten Lösungen insbesondere auch Steuerungsaufgaben detailliert erörtert, so daß die nachfolgende Diskussionen auch eine gute Basis für die Programmierung einer prozeduralen Ausführungssteuerung sind.

Die Entscheidung für einen triggerbasierten Ansatz ist auch durch das Ziel motiviert die Leistungsfähigkeit von bereits definierten SQL-Konzepten hinsichtlich funktionaler Erweiterungen von SQL aufzuzeigen. Für eine konkrete Implementierung ist jedoch das entscheidende Argument die Laufzeit der Prozesse während einer Transaktion und diese Entscheidung kann nur empirisch auf der Basis prototypischer Realisierungen 'rein aktiver', 'rein deduktiver' und kombinierter aktiver und deduktiver Ansätze entschieden werden. Die auszuführenden Aufgaben sind bei allen drei Lösungen die gleichen. Die Ansätze unterscheiden sich nur darin, ob sie in der Triggerkomponente des DB-Systems oder einer explizit programmierten Prozedur ausgeführt werden. Wie geeignet eine triggerbasierte Realisierung für praxisrelevante Anwendungen ist, hängt somit unmittelbar von der Implementierung der Triggerkomponente ab, u.a. davon wie effizient für ein Ereignis die Menge gefeuerter Trigger ermittelt werden kann und wie effizient große Mengen aktivierter Trigger verwaltet werden können.

Ceri et.al. empfehlen in [CCW00] aufgrund eben solcher Performanz-Probleme von Triggerausführungen gegenüber unmittelbar als DBS-Kernkomponente programmierten Funktionen einen Einsatz von Triggern in erster Linie für prototypische Implementierungen. Bei dieser Empfehlung muß beachtet werden, daß mit einer Realisierung mittels DBS-Kernprogrammen verglichen wird, die jedoch nur DBS-Herstellern möglich ist. Anwender von DB-Systemen hingegen müssen trotz der Performanz-Nachteile bei einer deduktiven 'bottom up'-Lösung die Schnittstelle 'dynamic SQL' verwenden.

5.1.3 Alternative triggerbasierte Ansätze

Die Eignung von BEFORE- und AFTER- bzw. ROW- und STATEMENT-Triggern für die Implementierung inkrementeller Verfahren wird in diesem Abschnitt diskutiert. Anhand von Lösungsskizzen werden die Möglichkeiten und Restriktionen der verschiedenen Triggertypen aufgezeigt.

Integritätsprüfung mittels BEFORE-Triggern

Für die Definition der IP-Trigger liegt die Verwendung von BEFORE-Triggern nahe, mittels derer eine Prüfung unmittelbar vor der Ausführung der Änderung durchgeführt wird (Abbildung 5.4). Auf diese Weise werden auch im Fehlerfall unnötige Aktionen vermieden, die zusätzlichen Aufwand beim Zurückrollen verursachen würden.

Änderungspropagierung mittels BEFORE-Triggern

Aufgrund des Ausführungsmodells der SQL-Trigger kann kein Propagierungsprozeß mittels BEFORE-Triggern implementiert werden. Zum einen sind in BEFORE-Triggern Ände-

rungsanweisungen nicht zugelassen, so daß von einem PR-BEFORE-Trigger keine Aktionen ausgeführt werden können, die die PR-Trigger nachfolgender Strata aktivieren könnten. Zum anderen kann eine zeitlich verzögerte Ausführung, wie sie für einen iterativen Fixpunktprozeß erforderlich ist, nicht realisiert werden. Abbildung 5.4 zeigt, daß zu jeder ausgeführten Änderungsanweisung ($*M, *M_1, *M_2$ mit $(* \in \{^+, ^-, ^\pm\})$) eine eigene BEFORE-Trigger-Abarbeitungsliste gehört, die vor der Basisfaktenänderung vollständig ausgeführt wird. EinBEFORE-Trigger läßt sich nicht in die Liste eines anderen Ereignisses einordnen.

Wird kein Änderungspropagierungsverfahren realisiert, dann ist eine Prüfung von auf Sichten definierten ASSERTIONs nur möglich, wenn die Ableitungsvorschriften der Sichten in die Formel der spezialisierten ASSERTION solange aufgefaltet wird, bis ausschließlich Basisrelationen referenziert werden. Ein Nachteil dieser Lösung ist die Restriktion, daß die Auffaltung nur für nicht rekursive Sichten durchführbar ist. Zudem kann sich die Laufzeit während einer Transaktion wesentlich verlängern, da ggf. aufgefaltete Formeln wiederholt angewendet werden. Im Rahmen der Schemaevolution tritt ebenfalls ein Mehraufwand auf, da für modifizierte Sichten nicht nur die Sichtdefinition angepaßt werden muß, sondern auch alle abhängigen IP-Trigger neu generiert werden müssen.

IMMEDIATE-Änderungspropagierung mittels AFTER-Triggern

Für die hier diskutierte Lösung kann der in Abschnitt 5.1.2 skizzierte triggerbasierte Implementierungsansatz zugrunde gelegt werden. Die dort eingeführten PR-Trigger werden dann, wie der PR-Trigger pr_1 im Beispiel 5.1.2, als AFTER-Trigger definiert. Da hier nur eine Lösung für ein Propagierungsverfahren erörtert wird, werden im weiteren SA- und IP-Trigger nicht berücksichtigt.

Wie gut der Zeitpunkt der Ausführung der direkt und indirekt durch eine Basisfaktenänderung $*M$ $(* \in \{^+, ^-, ^\pm\})$ gefeuerten AFTER-Trigger für die Durchführung der Änderungspropagierung und für die Simulation der IMMEDIATE-Integritätsprüfung geeignet ist, wird in Abbildung 5.4 deutlich. Die AFTER-Trigger (ta_1, ta_2) werden unmittelbar im Anschluß an die vom DB-System ausgeführte Integritätsprüfungsphase (IC_Check) ausgeführt. Mittels der Triggererzeugungszeiten (Prioritäten) muß sichergestellt werden, daß die PR-Trigger die ersten für $*M_i$ ausgeführten Trigger sind. Durch die Ausführung der PR- und auch der IP-Trigger vor den benutzerdefinierten Triggern wird sichergestellt, daß während des triggersimulierten Propagierungs- und Prüfungsprozesses der gleiche Zustand der Faktenbasis vorliegt wie bei der systeminternen IC_Check-Phase.

Abbildung 5.4 zeigt auch, daß die Simulation nur für unabhängige Basisfaktenänderungen wie $*M$, die keine triggerausgeführten $(*M_1, *M_2)$ und keine kaskadierenden Änderungen einer FOREIGN KEY-Bedingung sind, korrekt durchführbar ist. Da triggerausgeführte und kaskadierende Änderungen in diesem Zusammenhang völlig analog zu behandeln sind, werden sie im weiteren nur als triggerausgeführte bzw. als abhängige Basisfaktenänderungen bezeichnet.

Da alle AFTER-Trigger abhängiger Änderungen wie $*M_1, *M_2$ ebenfalls in die AFTER-Trigger-Liste der unabhängigen Änderung $*M$ gemäß ihrer Triggererzeugungszeit eingeordnet werden, wird eine zeitlich verzögerte Ausführung und damit die Ausführung eines iterativen Fixpunktprozesses möglich. Diese Eigenschaft des Triggerausführungsmodells

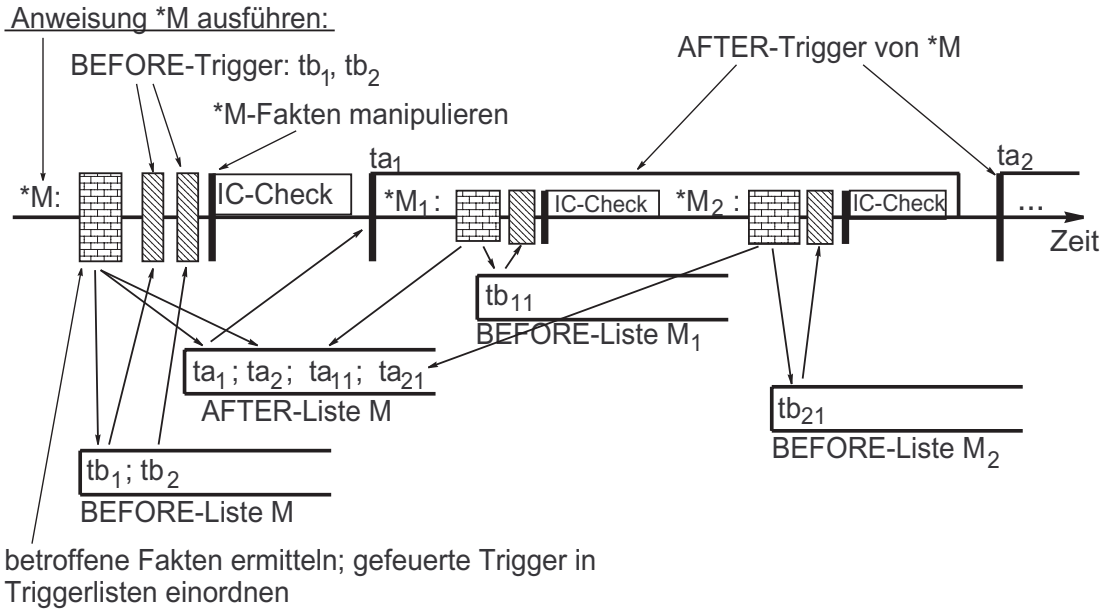

Abbildung 5.4: SQL-Integritätsprüfung und Trigger-Ausführung

verhindert aber zugleich, daß unmittelbar nach der Ausführung der abhängigen Änderungen $*M_1$ und $*M_2$ Trigger ausgeführt werden können. Die Trigger können frühestens nach Beendigung des Triggers ta_1 ausgeführt werden. Die systeminterne IC_Check-Phase hingegen wird unmittelbar nach $*M_1$ und $*M_2$ ausgeführt. Die Konsistenz der Änderung $*M_1$ kann mittels Trigger somit erst für einen Zustand nach Ausführung von $*M_2$ geprüft werden. Zur Lösung dieses Problems wird vorausgesetzt, daß im Aktionsteil benutzerdefinierter Trigger jeweils nur eine Änderungsanweisung programmiert ist. Dies stellt keine funktionale Einschränkung des SQL-Triggerkonzepts dar, da für einen Trigger mit mehreren Anweisungen mehrere Trigger mit je einer Anweisung definiert werden können. Eine entsprechende Sequentialisierung kann auch automatisch durchgeführt werden.

Die stratumbezogene Ausführungsreihenfolge der PR-Trigger wird durch ihre Erzeugungszeit gesteuert. Alle PR-Trigger, die im Stratum i ausgeführt werden sollen, haben eine frühere Erzeugungszeit als die des Stratums $i + 1$. Da alle PR-Trigger in direkter und indirekter Abhängigkeit von der ursprünglichen Basisfaktenänderung $*M$ feuern, werden sie gemäß ihrer Erzeugungszeit in die Abarbeitungsliste der AFTER-Trigger von $*M$ eingeordnet. Damit der Propagierungsprozeß vollständig vor den benutzerdefinierten Triggern ausgeführt wird, verfügen alle PR-Trigger über eine frühere Erzeugungszeit als die benutzerdefinierten. Diese Prioritätenzuordnung zusammen mit der Eigenschaft, daß nur eine Änderungsanweisung je Trigger zulässig ist, gewährleistet, daß auch für jede triggerausgeführte Änderungsanweisung unmittelbar anschließend ein Propagierungsprozeß gestartet wird. Die Prozeßtrigger werden dann vor dem nächsten benutzerdefinierten Trigger in die Liste eingeordnet. Daß der nächste benutzerdefinierte Trigger erst nach Beendigung des Propagierungsprozesses ausgeführt wird, ist völlig analog zu dem IMMEDIATE-Prüfungsverhalten in SQL.

Die Aufgabe der Ausführungsteuerung, die bei der gewählten Implementierungsstrategie von den Triggern übernommen werden soll, kann nur von AFTER-Triggern geleistet werden, denn das Ausführungsmodell der BEFORE-Trigger unterstützt die Steuerung der stratumbezogenen Durchführung eines Fixpunktprozesses nicht. Für die PR-Trigger werden somit im weiteren AFTER-Trigger verwendet. Welche Triggertypen für die IP- und SA-Trigger verwendet werden können, wird erst in den nachfolgenden Abschnitten bei der Definition der jeweiligen Verfahren erörtert.

ROW- vs. STATEMENT-Triggern:

Für die Ausführung einer Änderungsanweisung wird unabhängig von der Anzahl der manipulierten Fakten ein STATEMENT-Trigger aktiviert und ausgeführt (mengenorientiert). Da die geänderten Fakten während einer Triggerausführung in den systeminternen Transitionsrelationen vorliegen, kann auf das Lesen der Δ-Fakten in den Δ-Relationen verzichtet werden, wenn statt dessen auf die Transitionsrelationen zugegriffen wird. Von Vorteil ist dabei, daß z.B. in den Transitionsrelationen von PR-Triggern ausschließlich die zum aktuellen Zeitpunkt zu propagierenden Δ-Fakten vorliegen, während in den Δ-Relationen ggf. eine Menge von Δ-Fakten sowohl der aktuellen Änderungsanweisung als auch der vorangegangenen Änderungsanweisungen der Transaktion vorliegen kann. Bei Zugriffen auf Δ-Relationen müßten die aktuell relevanten Δ-Fakten explizit selektiert werden.

Werden ROW-Trigger gefeuert, dann wird für jedes manipulierte Fakt ein Trigger-Ereignispaar gebildet (instanzorientiert). Für die Trigger-Ereignispaare liegt das geänderte Fakt in den Transitionsvariablen vor. Auch bei ROW-Triggern kann der Zugriff auf die Δ-Relation durch den Zugriff auf die Transitionsvariablen ersetzt werden. Wie das Beispiel 5.1.3 zeigt, kann für STATEMENT-Trigger ein neues Relationssymbol p_NEW in der REFERENCING-Klausel der Transitionsrelation NEW RELATION des Triggers zugeordnet werden. Die Tupelvariable tv_p wird dann in der FROM-Klausel statt an p an das neue Relationssymbol p_NEW gebunden. Für ROW-Trigger kann die Tupelvariable tv_p, die in der Ableitungsvorschrift der Sicht s an die Relation p gebunden ist, in der REFERENCING-Klausel an die Transitionsvariable NEW ROW gebunden werden. Die Zuordnung des Relationssymbols p und der Tupelvariable tv_p kann aus der FROM-Klausel entfernt werden.

Beispiel 5.1.3 (ROW- und STATEMENT-Trigger)
Der PR-Trigger pr_1 aus Beispiel 5.1.2 kann wie folgt als STATEMENT- oder als ROW-Trigger modelliert werden.

TRIGGER pr_1 *AFTER INSERT ON* Δ^+p
REFERENCING NEW RELATION AS p_NEW *FOR EACH STATEMENT*
BEGIN ATOMIC INSERT INTO $\Delta^+s(a_1, a_2)$
SELECT $tv_p.a_1, tv_q.a_2$ *FROM* p_NEW *AS* tv_p, q *AS* tv_q
WHERE $tv_p.a_2 = tv_q.a_1$ *END;*

TRIGGER pr_1 *AFTER INSERT ON* Δ^+p
REFERENCING NEW ROW AS tv_p *FOR EACH ROW*
BEGIN ATOMIC INSERT INTO $\Delta^+s(a_1, a_2)$
SELECT $tv_p.a_1, tv_q.a_2$ *FROM* q *AS* tv_q
WHERE $tv_p.a_2 = tv_q.a_1$ *END.*

Da AFTER-Trigger unabhängig davon, ob sie als ROW- oder STATEMENT-Trigger definiert sind, in die gleiche AFTER-Liste eingeordnet werden, sind prinzipiell beide Triggerarten gleich gut geeignet. Die mengenorientierte Ausführung der STATEMENT-Trigger entspricht jedoch mehr einer reihenfolgeunabhängig durchgeführten Propagierung. Die Einordnung der ROW-Trigger-Ereignispaare ist aufgrund derselben Triggererzeugungszeit ebenfalls beliebig und nicht durch den Anwender steuerbar.

Entscheidendes Kriterium für die Verwendung von ROW-Triggern ist das unterschiedliche Verhalten beider Triggerarten, wenn durch eine Änderungsanweisung keine Fakten manipuliert werden. Wird während eines Propagierungsprozesses eine Änderungsanweisung ausgeführt, die keine Fakten manipuliert oder die Duplikate in eine Relation einfügt (effektlose Anweisung), dann feuert kein ROW-Trigger, und der Prozeß kann beschränkt auf die notwendigen Propagierungsschritte von der Triggerkomponente ohne zusätzliche Kontrolle gesteuert und beendet werden. Werden hingegen STATEMENT-Trigger verwendet, so werden ohne zusätzliche Abfragen der Inhalte von Transitionsrelationen immer alle direkt und indirekt von einer Änderungsanweisung abhängigen PR-Trigger ausgeführt, wodurch sich die Ausführungszeit des Prozesses ggf. wesentlich verlängern kann. Damit bei der Verwendung von STATEMENT-Triggern der Fixpunkt der Δ-Fakten einer rekursiven Regelmenge erkannt werden kann, müßte explizit ermittelt werden, welche Fakten in welcher Iterationsrunde abgeleitet wurden. Hingegen feuern ROW-Trigger automatisch nur für solche relevanten Fakten.

5.2 Inkrementelle Änderungspropagierung

In Abschnitt 5.2.1 wird ein Änderungspropagierungsverfahren im SQL-Kontext entwickelt und in Abschnitt 5.2.2 wird für dieses Verfahren ein triggerbasierter Implementierungsansatz vorgeschlagen. Wie Verfahren für die Sichtenaktualisierung und Integritätsprüfung integriert werden können, wird in den nachfolgenden Abschnitten erörtert. Um jedoch unnötige Wiederholungen zu vermeiden, werden bei einigen Diskussionen bereits die beiden Verwendungszwecke berücksichtigt.

5.2.1 Propagierungskonzept

Für das Änderungspropagierungsverfahren wird das DB-Schema $\mathcal{S}^{\mathcal{S}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C}, \mathcal{T} \rangle$ einer NSQL-Datenbank $\mathcal{D}^{\mathcal{S}}$ zugrunde gelegt. Die zugelassene Sichtenmenge $\mathcal{R}$ entspricht in Datalog einer stratifizierbaren Regelmenge. Rekursive Sichten und Sichtdefinitionen mit Negation sind zulässig. Aggregatfunktionen und Duplikate werden nicht berücksichtigt. Sie stellen keine SQL-spezifischen Anforderungen dar, und die in den Abschnitten 4.3.1 und 4.3.2 skizzierten Lösungen können mit geringfügigen Anpassungen integriert werden. Das in diesem Abschnitt entwickelte Propagierungsverfahren setzt voraus, daß nur virtuelle Sichten vorliegen. Die Optimierungsmöglichkeiten , die sich aus der Materialisierung abgeleiteter Faktenmengen ergeben, werden erst in Abschnitt 5.4 ausführlich erörtert. Das nachfolgende Propagierungsverfahren kann auch für materialisierte Sichten angewendet werden, sofern im Rahmen der Propagierung die materialisierten Sichten wie virtuelle behandelt werden, indem immer ihre Ableitungsvorschrift angewendet wird, ohne auf die gespeicherten Fakten zuzugreifen.

Da vorausgesetzt wird, daß die propagierten Δ-Fakten ggf. mehrfach verwendet werden, und zwar für die Ableitung weiterer induzierter Änderungen, für die Sichtenaktualisierung oder für die Integritätsprüfung, werden mit der Zielsetzung, den Aufwand bei der weiteren Verarbeitung zu minimieren, nur effektive Änderungen propagiert. Anderenfalls würden bei mehrfachen Verwendungen auch die unnötigen Δ-Fakten mehrfach verarbeitet. Die effektiven induzierten Änderungen werden im Rahmen eines änderungsgetriebenen Fixpunktprozesses ermittelt. Einen Eindruck von der triggerbasierten Implementierung und der 'bottom up'-Ausführung wurde bereits in Abbildung 5.3 vermittelt.

Die im weiteren erörterten zentralen Anforderungen an ein Propagierungsverfahren, wie die Propagierung zu zwei verschiedenen Zeitpunkten, die Nettoeffektberechnung und die Simulation zweier verschiedener Alt-Zustände resultieren unmittelbar aus dem SQL-Transaktions- und Integritätskonzept. Die Anforderung, auch Modifikationen zu propagieren, ist unabhängig vom Kontext.

IMMEDIATE- und DEFERRED Propagierung:

Das SQL-Integritätskonzept mit den beiden Prüfungszeitpunkten IMMEDIATE und DEFERRED erfordert ein angepaßtes Propagierungskonzept. Da IMMEDIATE-Integritätsbedingungen unmittelbar nach einer Basisfaktenänderung geprüft werden, ist es nicht ausreichend, wie in Datalog nur zum Transaktionsende zu propagieren. Werden hingegen alle induzierten Änderungen, auch die, die z.B. nur für die DEFERRED-Integritätsprüfung verwendet werden, unmittelbar nach einer Basisfaktenänderung propagiert, so liegen zum Transaktionsende zwar alle induzierten Änderungen vor, im Falle einer Konsistenzverletzung sind aber ggf. unnötig viele Propagierungsschritte durchgeführt worden. Diese IMMEDIATE propagierten, aber erst DEFERRED verwendeten Δ-Fakten stellen auch beim Zurückrollen zum letzten konsistenten Zustand einen unnötigen Mehraufwand dar. Da angestrebt wird, zu jedem der beiden Verarbeitungszeitpunkte nur den Aufwand an Zeit und Ressourcen zu verbrauchen, der zur Aufgabenerfüllung unbedingt erforderlich ist, wird der Änderungspropagierungsprozeß in einen IMMEDIATE- und einen DEFERRED-Prozeß aufgeteilt. Die beiden Zeitpunkte entsprechen denen des SQL-Integritätskonzepts.

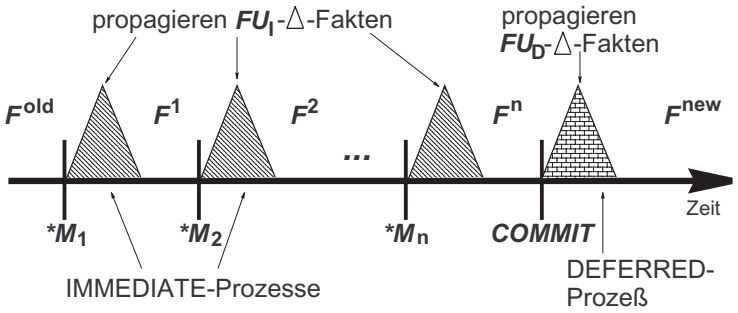

Abbildung 5.5: IMMEDIATE- und DEFERRED-Propagierung

Entscheidend für den Zeitpunkt der Propagierung von Δ-Fakten einer Relation ist der Zeitpunkt der frühestmöglichen Verwendung der induzierten Änderungen, der nicht allein durch den eigenen Verarbeitungszeitpunkt bestimmt wird sondern auch durch die Verar-

beitungszeitpunkte der direkt und indirekt abhängigen Sichten und Integritätsbedingungen. Für die nachfolgende Klassifizierung der Δ-Relationen wird sowohl eine IMMEDIATE und DEFERRED durchgeführte Integritätsprüfung vorausgesetzt wie auch eine zu beiden Zeitpunkten durchgeführte Sichtenaktualisierung. Das Integritätskonzept ist das in Abschnitt 3.1 vorgestellte. Das Sichtkonzept wird in Abschnitt 5.4 definiert. Der verwendete Begriff des DB-Objekts bezieht sich somit auf Sichten und Integritätsbedingungen.

Sei $R \in \mathcal{R} \cup \mathcal{B}$ die Definition einer materialisierten oder virtuellen Sicht oder eine Basisrelation mit den zugehörigen relevanten Δ^*-Relationen ($* \in \{^+, ^-, ^\pm\}$) eines DB-Schemas $\mathcal{S}^{\mathcal{S}}$. Die Δ^*-Relationen werden in zwei disjunkte Klassen ($\mathcal{FU}_I \cap \mathcal{FU}_D = \emptyset$) unterteilt.

- $\Delta^* R$ ist eine 'first used' IMMEDIATE-Δ-Relation (FU_I-Relation), gdw. $rel(R)$ das Symbol einer materialisierten Sicht ist und IMMEDIATE aktualisiert wird, oder mindestens ein IMMEDIATE-DB-Objekt von R abhängt.

- $\Delta^* R$ ist eine 'first used' DEFERRED-Δ-Relation (FU_D-Relation), gdw. $rel(R)$ das Symbol einer materialisierten Sicht ist und DEFERRED aktualisiert wird, oder ausschließlich DEFERRED-DB-Objekte von R abhängen.

$\mathcal{FU}_I/\mathcal{FU}_D$ sind die Mengen aller FU_I-/FU_D-Relationen eines DB-Schemas. Während eines IMMEDIATE-/DEFERRED-Prozesses werden jeweils FU_I-/FU_D-Δ-Fakten propagiert. Der IMMEDIATE-Prozeß beginnt mit der Propagierung der Δ-Fakten der aktuell auszuführenden Basisfaktenänderungen und wird beendet, sobald alle FU_I-Δ-Fakten vorliegen. Nach Beendigung des IMMEDIATE-Prozesses können mit dem Ziel der Einsparung von Speicherplatz die für den DEFERRED-Prozeß nicht mehr benötigten Δ-Fakten gelöscht werden. Nicht mehr benötigt werden die Δ-Fakten, von deren Relationen ausschließlich IMMEDIATE-DB-Objekte direkt abhängen. Besteht für mindestens ein DEFERRED-DB-Objekt eine direkte Abhängigkeit, so wird dieses Δ-Fakt bis zum DEFERRED-Propagierungsprozeß gespeichert. Für die Ableitung der FU_D-Δ-Fakten werden die FU_I-Δ-Fakten zugrunde gelegt. Somit wird vermieden, daß Fakten, auf die in beiden Prozessen zugegriffen werden, zweimal abgeleitet werden. Zur Identifizierung dieser Δ-Fakten, die nur im IMMEDIATE-Prozeß benötigt werden, wird die Klasse der 'last used'-IMMEDIATE-Δ-Fakten gebildet, die eine echte Teilmenge der FU_I-Δ-Fakten ist ($\mathcal{LU}_I \subseteq \mathcal{FU}_I$).

- ΔR ist eine 'last used' IMMEDIATE-Δ-Relation (LU_I-Relation), gdw. $\Delta R \in \mathcal{FU}_I^\Delta$ und keine DEFERRED-DB-Objekte von R abhängen. $\mathcal{LU}_I$ ist die Menge aller LU_I-Relationen eines DB-Schemas.

Für das Beispiel 3.1.4 aus Abschnitt 2.2 veranschaulicht Abbildung 5.6, zu welchem Zeitpunkt welche Fakten propagiert werden. Während des IMMEDIATE-Prozesses werden die Δ-Fakten der FU_I-Relationen *mitarbeiter*, *abteilung*, *mita_abt*, *mita_mgr* propagiert. Die Propagierung des DEFERRED-Prozesses startet mit den Δ-Fakten der FU_I-Relationen, die keine LU_I-Relationen sind (*mitarbeiter*, *mita_mgr*), und auf ihrer Basis werden die Δ-Fakten der FU_D-Relation *lower_mgr* propagiert.

Modifikationen:

Neben Einfügungen und Löschungen werden auch Modifikationen propagiert. Bereits im Datalog-Kontext (Abschnitt 4.3.3) wurde ein gegenüber Urpí/Olivé ([UO92]) und

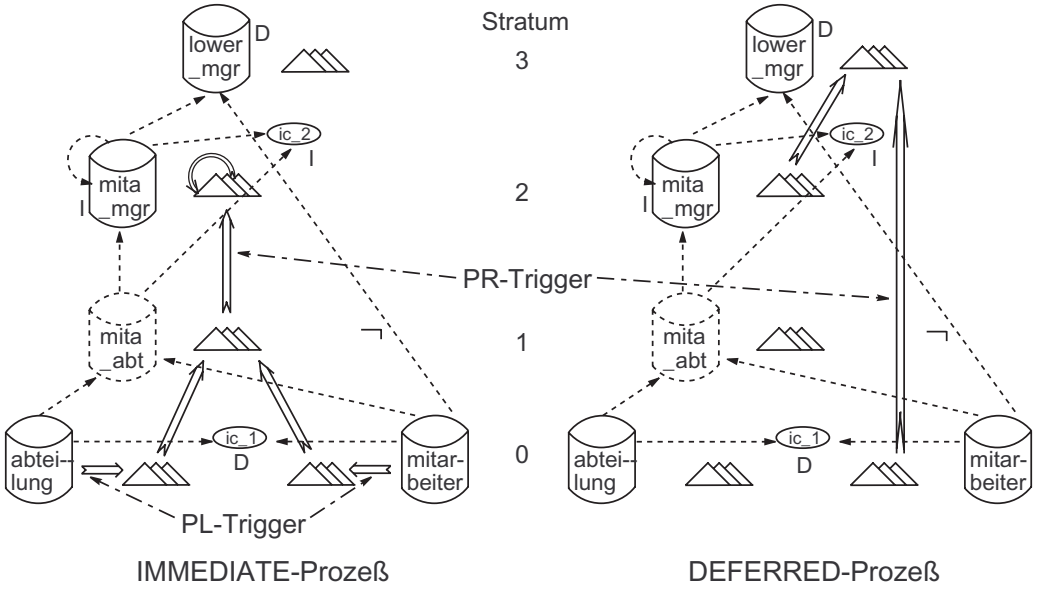

Abbildung 5.6: IMMEDIATE- und DEFERRED-'bottom up'-Propagierung

Mayol/Teniente ([MT95]) erweiterter Ansatz zur Propagierung von Modifikationen entwickelt. Das dort spezifizierte Konzept muß der Syntax von NSQL angepaßt werden.

Sei $S \in \mathcal{R}$ eine Sichtdefinition von $\mathcal{D}^{\mathcal{S}}$ und $R_1, ..., R_m$ die Relationen, von denen S direkt abhängig ist ($S <_1 R_i$, $1 \leq i \leq m$), wobei $R_1, ..., R_k$ ($k \leq m$) in der FROM-Klausel des Anfrageausdrucks auftreten und $R_{k+1}, ..., R_m$ in Unteranfragen.

- $\pm R_i$ induziert eine Modifikation ($\pm S$),
  gdw. $R_i \in \{R_1, ..., R_k\} \wedge$ sich ausschließlich Werte von R_i-Attributen ändern, die in der SELECT-Liste auftreten, aber nicht in der WHERE-Klausel des Anfrageausdrucks oder in einer Unteranfrage.

- $\pm R_i$ induziert eine Einfügung ($+S$) und Löschung ($-S$),
  gdw. $R_i \in \{R_1, ..., R_m\} \wedge R_i$-Attribute manipuliert werden, die in der WHERE-Klausel des Anfrageausdrucks oder in einer Unteranfrage auftreten.

- $\pm R_i$ wird nicht propagiert,
  gdw. $R_i \in \{R_1, ..., R_m\} \wedge$ ausschließlich R_i-Attribute modifiziert werden, die weder in der SELECT-Liste noch in der WHERE-Klausel des Anfrageausdrucks oder in einer Unteranfrage auftreten.

Diese attributbezogene Relevanz gilt unter Berücksichtigung der Polarität analog für Integritätsbedingungen. Ist eine Integritätsbedingung IC relevant für eine Modifikation von R_i, und ist sie auch von einer Löschung von R_i-Fakten betroffen, dann wird IC für die alten Werte des manipulierten Fakts geprüft. Ist IC für $\pm R_i$ und $+R_i$ relevant, dann wird IC für die neuen Werte geprüft.

Nettoeffektberechnung

Da während eines IMMEDIATE-Prozesses eine Menge von Basisfaktenänderungen der aktuellen Änderungsanweisung propagiert wird und während des DEFERRED-Prozesses die Folge der Basisfakten- und induzierten Änderungen der aktuellen Transaktion, stellt sich die Frage nach der Berechnung des Nettoeffekts der Änderungen (Abschnitte 2.1.3, 2.2.4). Aufgrund des SQL-Transaktionskonzepts ergeben sich für Basisfaktenänderungen während eines IMMEDIATE- und eines DEFERRED-Prozesses ggf. unterschiedliche Bedeutungen.

Die Semantik einer Basisfaktenänderung und der durch sie induzierten Änderungen in einem IMMEDIATE-Propagierungsprozeß wird durch verschiedene Eigenschaften bestimmt. Eine zentrale Eigenschaft ist neben der Ausführung nur effektiver Basisfaktenänderungen (Definition 2.2.18) auch die Propagierung nur effektiver induzierter Änderungen. Stellt z.B. eine Basisfaktenänderung die Einfügung eines Duplikats dar oder die Löschung eines nicht vorhandenen Fakts, so wird sie weder ausgeführt noch propagiert (effektlose Änderung). Diese beiden Eigenschaften zusammen mit den Tatsachen, daß die Änderungen einer Anweisung konfliktfrei sind, und, daß im Rahmen der IMMEDIATE-Propagierung nur Änderungen einer Anweisung verarbeitet werden, gewährleisten, daß keine Konflikte auftreten können. Eine Nettoeffektberechnung für eine IMMEDIATE-Propagierung ist somit nicht erforderlich.

Während eines IMMEDIATE-Propagierungsprozesses werden die Änderungen der aktuellen Anweisung völlig isoliert betrachtet. Alle Änderungen vorangegangener Anweisungen der Transaktion werden nicht berücksichtigt, so daß z.B. die Löschung eines Fakts, das erst in der aktuellen Transaktion eingefügt wurde, während des IMMEDIATE-Prozesses behandelt wird, als wäre das Fakt bereits zu Beginn der Transaktion vorhanden gewesen. Während eines DEFERRED-Prozesses heben sich die beiden kompensierenden Änderungen (Einfügung und Löschung des gleichen Fakts) in ihrer Wirkung auf und werden weder propagiert noch verarbeitet. Zum Ende einer Transaktion muß also sehr wohl ein Nettoeffekt berechnet werden.

Da nur effektive Änderungen ausgeführt und propagiert werden, treten in einem DEFERRED-Prozeß nur die in Abschnitt 2.1.3 erörterten Konfliktfälle 2, 3, 6 und 7 auf. Da aber eine SQL-Transaktion eine Folge von Änderungen ist, stellen diese vier Fälle keine Konflikte dar, sondern lediglich Änderungssequenzen (Folgemodifikationen), die zu einer Änderung zusammengefaßt werden können. So kann z.B. die Sequenz $+p(a,b); \pm(p(a,c)p(a,b))$ zu der Änderung $+p(a,c)$ zusammengefaßt werden. Bei der Sequenz $+p(a,b); -p(a,b)$ heben sich die Änderungen in ihrer Wirkung auf und werden nicht propagiert (effektlose Änderung).

Von den DB-Systemen kommerzieller Hersteller werden zwar die Nettoeffektfakten zum Ende der Transaktion z.B. für die Integritätsprüfung verwendet, sie sind aber für Anwender nicht zugreifbar. Auch das SQL-Triggerkonzept bietet, da zum Transaktionsende keine Trigger gefeuert oder ausgeführt werden, keine Möglichkeiten, auf die Nettoeffektfakten zuzugreifen. Es ist somit Aufgabe des Propagierungsverfahrens, die Nettoeffektfakten bereitzustellen. Zu diesem Zweck werden die Δ-Fakten der Basisfaktenänderungen und der durch sie induzierten Änderungen in Δ-Basisrelationen gespeichert.

Zur Identifizierung der Reihenfolge des Auftretens der Änderungen in einer Transaktion wird für eine Änderungsanweisung und den durch sie manipulierten Basisfaktenänderungen sowie ihren induzierten Änderungen eine **Propagierungsprozeßnummer (PP-Nummer)** als ein gemeinsamer Zeitstempel vergeben. Die PP-Nummer ist innerhalb einer Transaktion eindeutig und kann ein Zeitstempel sein oder die Nummer der Ausführungsreihenfolge der Änderungsanweisung. Mittels der PP-Nummer kann dann zum Transaktionsende die für die Nettoeffektberechnung entscheidende Reihenfolge der Änderungen ermittelt werden. Die Struktur jeder Δ-Relation wird um die PP-Nummer *pp_id* ergänzt. Sei $R \in \mathcal{R} \cup \mathcal{B}$ eine Relation mit n Attributen, dann werden vom Präprozessor Δ-Relationen mit folgendem Schema generiert:

- Für Einfügungen/Löschungen: $\Delta^{\star}R \ \ (A_1, ..., A_n, \ pp\_id)$ mit $\star \in \{^+, ^- \}$.
- Für Modifikationen: $\Delta^{\pm}R \ \ (A_1^{new}, ..., A_n^{new}, A_1^{old}, ..., A_n^{old}, \ pp\_id)$
  $\Delta^{\pm +}R(A_1^{new}, ..., A_n^{new}, \ pp\_id)$
  $\Delta^{\pm -}R(A_1^{old}, ..., A_n^{old}, \ pp\_id)$.

Der Präprozessor verwaltet zur Vergabe der korrekten PP-Nummer transaktionsspezifisch eine globale Variable, die sogenannte SYSTEM_PP_ID.

Zudem wird die PP-Nummer für die **Simulation des alten Zustands** R^{old} einer Relation R verwendet. Aufgrund des SQL-Transaktionskonzepts müssen die lokal- und global konsistenten Zustände unterschieden werden. Im Rahmen eines IMMEDIATE-Prozesses bezeichnet R^{old} den lokal-konsistenten Zustand vor Beginn der Ausführung der Basisfaktenänderung $*M_i$ aus der Transaktion $\mathcal{M} := \{*M_1; ...; *M_{i-1}; *M_i; ...; *M_n\}$. Die für die Simulation von R_i^{old} notwendige Unterscheidung der Δ-Fakten einer Relation, ob sie durch die aktuelle $(*M_i)$ oder vorangegangene Basisfaktenänderungen $(*M_1, ..., *M_{i-1})$ induziert wurden, wird anhand der PP-Nummer getroffen. Ausführlich wird die Simulation der verschiedenen alten Zustände zusammen mit den Transitionssichten erörtert (Definition 5.2.3).

Zwei weitere naheliegende Aufgabenstellungen einer solchen PP-Nummer werden durch die Triggerkomponente gelöst. Da in den Δ-Relationen alle Fakten seit Beginn der Transaktion gespeichert sind, müßte bei einer deduktiven Implementierung mit Hilfe der PP-Nummer selektierend auf die Δ-Relationen zugegriffen werden, um nur die für den aktuellen Prozeß zu propagierenden Fakten zu ermitteln. Bei der aktiven Implementierungsstrategie hingegen liefert das feuernde Ereignis, die Einfügung eines Δ-Fakts, die notwendigen Informationen. Ein weiterer Punkt ist die Simulation des lokalen ROLLBACKs. Bei einer deduktiven Implementierung müßten alle Δ-Fakten mit der gleichen PP-Nummer wie die fehlererzeugende Basisfaktenänderung explizit aus den Δ-Relationen gelöscht werden. Bei der aktiven Implementierungsstrategie wird der gleiche Effekt durch den Abbruch (fehlerhaftes Beenden) der Ausführung eines Triggers erreicht.

Präprozessor-Lösung:

Für die Ausführung des Präprozessors zur Schemaentwurfszeit können drei Hauptaufgaben identifiziert werden. Zur ersten Aufgabe der Analyse des DB-Schemas gehört u.a. die FU-, LU-Klassifizierung. Ziel dieser und weiterer noch zu diskutierender Analyseaufgaben ist es, die Triggerausführungen soweit wie möglich auf das Minimum zu beschränken.

Diese Phase beinhaltet auch die Speicherung der Analyse-Ergebnisse im Präprozessor-Schema. Ausgehend von den im DB- und Präprozessor-Schema dokumentierten Basisrelationen, Sichten und ASSERTIONs werden die für die Verfahrensausführung erforderlichen DB-Objekte wie Δ-Basisrelationen, Transitionssichten (2. Aufgabe) und Prozeßtrigger (3. Aufgabe) generiert. Welche der Triggerarten (PL, PR, IP oder SA) jeweils zu den Prozeßtriggern gehören, hängt von der Aufgabenstellung des Verfahrens ab.

BEGIN
  Analysieren des DB-Schemas $\mathcal{S}^{\mathcal{S}}$ der Datenbank $\mathcal{D}^{\mathcal{S}}$;
  Generieren der relevanten Δ-Basisrelationen und Transitionssichten;
  Generieren der relevanten Prozeßtrigger;
END.

Da virtuelle Sichten auch in kommerziellen SQL-basierten DB-Systemen zulässige DB-Objekte sind, kann der Präprozessor unmittelbar auf die systeminternen Schemarelationen zugreifen. Zur Verwaltung der Sichten können die entsprechenden SQL-DDL-Anweisungen verwendet werden. Da keine Trigger für DDL-Anweisungen definiert werden können, muß der Anwender zum Ende des Schemaentwurfs bzw. zum Ende einer Schemamodifikation die Präprozessor-Prozedur aufrufen.

5.2.2 Triggerbasierte Propagierung

Aufgrund der Komplexität verschiedener Teilprobleme wird die Implementierungsstrategie in vier Themenschwerpunkten ausführlich erörtert. Zuerst wird die 'bottom up'-Propagierung induzierter Änderungen diskutiert, wie sie gleichermaßen für den IMMEDIATE- wie auch DEFERRED-Prozeß durchgeführt wird. Der für beide Prozesse sehr unterschiedliche Start wird in zwei separaten Themenblöcken erörtert. Abschließend werden die Transitionssichten und die spezialisierten Ableitungsvorschriften, die als Anfragen im Triggeraktionsteil auftreten, definiert.

'Bottom up'-Propagierung:

Statt der deduktiven Δ-Regeln beim generischen Verfahren werden sogenannte Propagierungstrigger (PR-Trigger) mit den beiden Aufgaben der Propagierung induzierter Änderungen und der Steuerung der 'bottom up'-Änderungspropagierung generiert (Beispiel 5.1.2). Analog zu dem Vorgehen bei der Generierung spezialisierter Datalog-Regeln werden für jede Relation $R \in \mathcal{RV} \cup \mathcal{B}$, die in einer Sichtdefinition $S \in \mathcal{RV}$ auftritt, und für jede relevante Änderungsart ($* \in \{^+, ^-, ^\pm\}$) PR-Trigger generiert. Im Aktionsteil tritt die Formel der spezialisierten Ableitungsvorschrift von S als Anfrageausdruck auf. PR-Trigger feuern für die Einfügung eines Δ-Fakts in eine Δ-Relation. Bei der Triggerausführung wird die Formel der spezialisierten Ableitungsvorschrift angewendet, um die induzierten Δ-Fakten abzuleiten, die dann in die entsprechende Δ-Relation von S eingefügt werden, wodurch die zugehörigen PR-Trigger aktiviert werden.

Zur Steuerung der Ausführung der PR-Trigger zum IMMEDIATE- oder DEFERRED-Zeitpunkt erzeugt der Präprozessor im DB-Schema eine Basisrelation '**propagation**' mit dem Attribut '*mode*'. Transaktionsspezifisch wird dort das Kennzeichen 'I' bzw. 'D' für den aktuell bearbeiteten Prozeß gespeichert. Der Präprozessor generiert IMMEDIATE- und DEFERRED-PR-Trigger, in deren WHEN-Klausel das **propagation**-Kennzeichen abgefragt

wird. Nachteil dieser Lösung ist, daß alle PR-Trigger zu einem relevanten Ereignis feuern und in die Ausführungsliste sowohl Trigger-Ereignispaare von IMMEDIATE- wie auch DEFERRED-Triggern eingeordnet werden. Erst zum Ausführungszeitpunkt wird die Bedingung geprüft, so daß nur die relevanten Prozeßtrigger ausgeführt werden.

Die stratumbezogene **Ausführungsreihenfolge** der PR-Trigger wird durch den Zeitpunkt ihrer Generierung gesteuert. Die Erzeugungszeit der PR-Trigger, die im Stratum j ausgeführt werden sollen, ist kleiner als die der PR-Trigger des Stratums $j + x$ ($x > 0$). Die PR-Trigger werden unabhängig von ihrem Aktivierungszeitpunkt gemäß ihrer Triggererzeugungszeit in die Abarbeitungsliste der AFTER-Trigger eingeordnet und somit stratumbezogen ausgeführt (Abschnitt 5.1.3, Abbildung 5.4). Der Propagierungsprozeß **terminiert** automatisch, wenn mittels PR-Triggern keine neuen Δ-Fakten mehr abgeleitet werden können, oder wenn keine neuen PR-Trigger gefeuert werden. Die Zusammenhänge zwischen der Einordnung in die AFTER-Liste einer Basisfaktenänderung und dem Ausführungszeitpunkt wird in den Abbildungen 5.7 und 5.8 skizziert.

Das Konzept der Einordnung in die AFTER-Liste gewährleistet auch die korrekte Propagierung von Δ-Fakten **rekursiver** Sichten. Alle Trigger rekursiv abhängiger Sichten gehören einem Stratum an, und mit der Bearbeitung des nächsten Stratums kann erst fortgefahren werden, wenn alle Trigger-Ereignispaare des aktuellen ausgeführt sind und keine neuen mehr eingeordnet werden. Aufgrund des Triggerausführungsmodells und des Verbots von Duplikaten werden wiederholte Ableitungen von Δ-Fakten erkannt und nicht weiterpropagiert. Denn eine Änderung, die ein Duplikat verursachen würde, ist eine effektlose Änderung, und für effektlose Änderungen werden keine AFTER-ROW-Trigger gefeuert.

Da das feuernde Ereignis die Einfügung in eine **Δ-Relation** ist, und die Trigger nur für Basisrelationen definiert werden können, werden alle Δ-Relationen als Basisrelationen definiert. Die Ausführung von Änderungsanweisungen auf Relationen ist als Ereignis für die PR-Trigger ungeeignet. Zwar könnten Trigger für die Basisrelationen, mit denen materialisierte Sichten simuliert werden, definiert werden jedoch für virtuelle Sichten können die Trigger nicht formuliert werden. Zudem können Änderungsanweisungen nur für bestimmte Sichten ausgeführt werden. Aufgrund des triggerbasierten Implementierungsansatzes sind Optimierungsstrategien ausgeschlossen, die versuchen, die Ausführungszeit des Propagierungsprozesses unter Abwägung der Kosten für die Speicherung und der Kosten für ggf. wiederholte Ableitungen weiter zu reduzieren, indem individuell für jede Δ-Relation entschieden wird, ob sie als Basisrelation oder als Sicht realisiert wird (Abschnitt 4.3.7).

Für eine übersichtlichere Darstellung wird die Definition der **PR-Trigger** in die des Triggers und in die der spezialisierten Δ-Anfrage (Definition 5.2.2) unterteilt. Ein Beispiel für einen PR-Trigger wird zusammen mit der zugehörigen Δ-Anfrage in Beispiel 5.2.1 gegeben. Um über die Relevanz von Sichten im Rahmen eines Propagierungsprozesses zur Schemaentwurfszeit entscheiden zu können, werden vom Präprozessor die Abhängigkeitsgraphen für Relationen G_D (Definition 2.1.12) und für Attribute G_D^{attr} (Definition 2.1.13) für das zugrunde liegende DB-Schema $\mathcal{S}^{\mathcal{S}}$ erstellt. Die Beurteilung der Relevanz eines PR-Triggers sowie die Berücksichtigung der Polaritäten von zu propagierenden und induzierten Änderungen erfolgt bereits bei der Generierung der spezialisierten $\Delta^{\dagger}S_{\Delta^{\ddagger}R}$-*query*-Anfrage ($\dagger, \ddagger \in \{^{+}, ^{-}, ^{\pm}\}$) , so daß für jede generierte Δ-Anfrage ein ent-

sprechender PR-Trigger generiert werden muß.

CREATE TRIGGER $pr\_\Delta^{\dagger}S_{\Delta^{\ddagger}R}$ AFTER INSERT ON $\Delta^{\ddagger}R$
REFERENCING NEW ROW AS T_R FOR EACH ROW
WHEN (EXISTS (SELECT 1 FROM *propagation* WHERE *mode* = 'I' | 'D'))
BEGIN ATOMIC INSERT INTO $\Delta^{\dagger}S$ $\Delta^{\dagger}S_{\Delta^{\ddagger}R}\_query$; END;

PR-Trigger feuern für die Ausführung einer Einfügung in die $\Delta^{\ddagger}$-Basisrelation der Relation R. An die Transitionsvariable wird die Tupelvariable T_R gebunden, die in der Ableitungsvorschrift von S an die Relation R gebunden ist und als freie Tupelvariable in der $\Delta^{\dagger}S_{\Delta^{\ddagger}R}\_query$-Anfrage auftritt. Mittels der WHEN-Formel wird die Ausführung dieses Trigger während eines IMMEDIATE- oder DEFFERED-Prozesses gesteuert. Als Aktion wird die zur Laufzeit mit den Werten des eingefügten Δ-Fakts instanziierte $\Delta^{\dagger}S_{\Delta^{\ddagger}R}\_query$-Anfrage angewendet. Die Ergebnisfakten werden in die $\Delta^{\dagger}S$-Basisrelation eingefügt.

Bei der Propagierung einer **Modifikation** $\pm R$ darf ein PR-Trigger ausschließlich für Wertänderungen relevanter Attribute feuern. Die Wertänderungen müssen explizit in der WHEN-Bedingung oder im Aktionsteil abgefragt werden. Seien $A_g, ..., A_h$ die Attribute von R, deren Wertänderungen relevant für induzierte Einfügungen/Löschungen sind, und $A_k, ..., A_l$ ($\{A_g, ..., A_h\} \cap \{A_k, ..., A_l\} = \emptyset$) die, die für induzierte Modifikationen relevant sind. Dann kann die WHEN-Bedingung für eine attributbezogene Ausführungskontrolle wie folgt erweitert werden:[4]

für induzierte Einfügungen $+S$/Löschungen $-S$:
AND ($T_R.A_g^{new} \neq T_R.A_g^{old}$ OR ... OR $T_R.A_h^{new} \neq T_R.A_h^{old}$),

für induzierte Modifikationen $\pm S$:
AND ($T_R.A_k^{new} \neq T_R.A_k^{old}$ OR ... OR $T_R.A_l^{new} \neq T_R.A_l^{old}$)
AND ($T_R.A_g^{new} = T_R.A_g^{old}$ AND ... AND $T_R.A_h^{new} = T_R.A_h^{old}$).

Präprozessor-Lösung:

Damit die für eine Relevanzanalyse erforderlichen Informationen für die Triggergenerierung zur Verfügung stehen, werden vom Präprozessor Abhängigkeitsgraphen erstellt. Der Schritt der Triggergenerierung kann nun hinsichtlich des Zeitpunkts der Erzeugung der einzelnen PR-Trigger differenzierter formuliert werden. Sei $\mathcal{RV}$ eine Menge virtueller Sichten und λ_{max} das maximale Stratum einer Sichtenmenge.

Alle generierten DB-Objekte werden im DB-Schema erzeugt, so daß die Kontrolle ihrer Ausführung dem DB-System unterliegt. Zur Speicherung der für die Relevanzanalyse erforderlichen Informationen wird das Präprozessor-Schema um Relationen für die Abhängigkeitsgraphen und die Stratazuordnung erweitert.

[4] Obwohl im Hinblick auf die Transformation zwischen Datalog und NSQL der Disjunktionsoperator nicht in NSQL-Formeln zulässig ist, wird er hier aus Gründen der besseren Lesbarkeit trotzdem verwendet. Ist kein OR-Operator zulässig, so muß obige Bedingung im Aktionsteil als Selektion bzgl. der Transitionstabellen mit entsprechenden UNION-Verknüpfungen formuliert werden.

BEGIN

−− Analysieren des DB-Schemas $\mathcal{S}^{\mathcal{S}}$ der Datenbank $\mathcal{D}^{\mathcal{S}}$;
Erstellen der Abhängigkeitsgraphen G_D und G_D^{Attr};
Klassifizieren FU-, LU-Δ-Relationen und -Integritätsbedingungen;
$\forall\ R \in \mathcal{B} \cup \mathcal{RV}$: Zuordnen der Strata $\lambda(R)$;

−− Generieren der Δ-Basisrelationen und Transitionssichten;
$\forall\ R \in \mathcal{B} \cup \mathcal{RV}$: Generieren der relevanten $\Delta^* R$-Basisrelationen $(* \in \{^+, ^-, ^\pm\})$

−− Generieren der Prozeßtrigger;
$j := 1$;
REPEAT
  $\forall\ R \in \mathcal{RV} \wedge \lambda(R) = j$: Generieren der relevanten PR-Trigger;
  $j := j + 1$;
UNTIL $j = \lambda_{max}(\mathcal{RV})$;

END.

Start eines IMMEDIATE-Prozesses:

Die Ausführung eines IMMEDIATE-Prozesses unmittelbar nach der Ausführung einer Änderungsanweisung $*M \in \mathcal{M}$ $(* \in \{^+, ^-, ^\pm\})$ wird durch die Ausführung von $*M$ selbst gestartet. Da aber PR-Trigger für Δ-Relationen definiert sind, werden zuerst sogenannte Protokolltrigger (PL-Trigger) ausgeführt, die die Δ-Fakten der Basisfaktenänderungen zusammen mit der aktuellen PP-Nummer (SYSTEM_PP_ID) in die zugehörige Δ-Relation speichern. Dieses Protokollereignis feuert dann die PR-Trigger des ersten Stratums. Die Erzeugungszeit jedes PL-Triggers ist kleiner als die aller PR-Trigger. Da alle Prozeßtrigger (PL- und PR-Trigger) direkt oder indirekt durch $*M$ aktiviert werden, werden alle gemäß ihrer Erzeugungszeit in die AFTER-Triggerliste von $*M$ eingeordnet und stellen für das DB-System einen gemeinsamen Prozeß dar. Diese mittelbare und unmittelbare Abhängigkeit aller Trigger von einem ursprünglichen Ereignis ermöglicht erst eine stratumbezogene Ausführung der Trigger. Zudem ist die Abhängigkeit wichtig zur Simulation eines lokalen ROLLBACKs bei einer Konsistenzverletzung (Abschnitt 5.3). Die Einordnung der Prozeßtrigger in eine gemeinsame AFTER-Liste und ihre stratumbezogene Ausführung wird in Abbildung 5.7 veranschaulicht.

Der Präprozessor generiert für jede relevante Basisrelation $B \in \mathcal{B}$ und für jede zugehörige Δ-Basisrelation $\Delta^\dagger B$ $(\dagger \in \{^+, ^-, ^\pm\})$ einen PL-Trigger $pl\_\Delta^\dagger B$ als AFTER ROW-Trigger. Die Tupelvariable für das eingefügte Fakt bzw. für die neuen Werte des modifizierten Fakts ist T^{new} und die für das gelöschte bzw. für die alten des modifizierten Fakts ist T^{old} $(\Phi \in \{T^{new}, T^{old}\})$. Die globale Variable SYSTEM_PP_ID enthält die eindeutige PP-Nummer der aktuell bearbeiteten Änderungsanweisung, damit alle zugehörigen Basisfakten- und induzierten Änderungen identifiziert werden können.

CREATE TRIGGER $pl\_\Delta^\dagger B$
AFTER INSERT | DELETE | UPDATE OF $(A_g, ..., A_h)$ ON B
REFERENCING OLD ROW AS T^{old} NEW ROW AS T^{new} FOR EACH ROW
BEGIN ATOMIC INSERT INTO $\Delta^\dagger B$
  VALUES ($\Phi.A_1, ..., \Phi.A_n$, SYSTEM_PP_ID) ;
END;

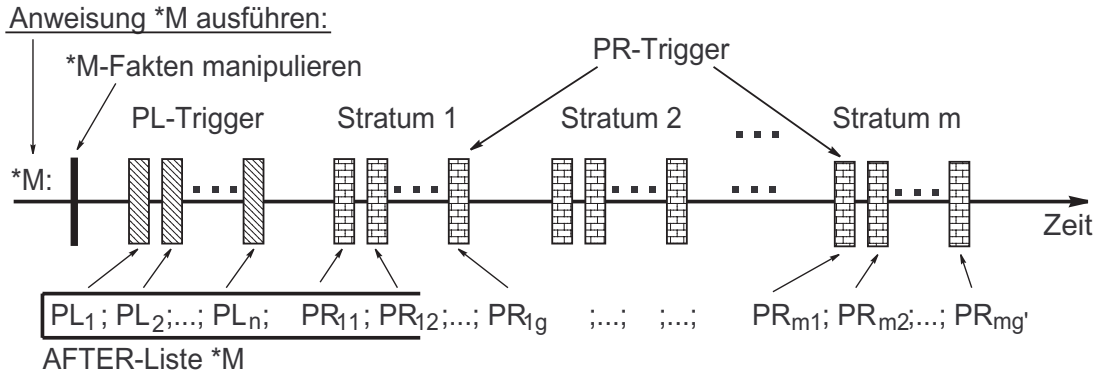

Abbildung 5.7: Triggerreihenfolge der IMMEDIATE-Propagierung

Die Attribute $A_g, ..., A_h$ sind diejenigen Attribute, deren Wertänderungen für Sichten und Integritätsbedingungen relevant sind. Bei einer Modifikation muß obige Einfügeanweisung gemäß der Attributstruktur der $\Delta^{\pm}$-Basisrelation angepaßt werden.

Präprozessor-Lösung:

Damit der Start von IMMEDIATE-Prozessen gewährleistet ist, wird die Aufgabe der Generierung von Prozeßtriggern um die Erzeugung der PL-Trigger erweitert.

$--$ Generieren der Prozeßtrigger;
$\forall\ B \in \mathcal{B}:$ Generieren der relevanten PL-Trigger;
...

Dieser Schritt ist vor der Generierung der PR-Trigger auszuführen, damit die Triggererzeugungszeiten der PL-Trigger kleiner sind als die der PR-Trigger.

Start eines DEFERRED-Prozesses:

Der Start eines DEFERRED-Prozesses zum Transaktionsende ist wesentlich komplexer als der eines IMMEDIATE-Prozesses, da zum Transaktionsende keine Trigger gefeuert oder ausgeführt werden. Der DEFERRED-Prozeß wird durch den Aufruf einer DB-Prozedur *commit_iv*[5] gestartet, die vom Anwender anstelle der COMMIT-Anweisung aufgerufen wird. Während einer Transaktion ist dieser Prozeduraufruf der einzige Punkt, an dem die Systemerweiterung für den Anwender nicht transparent ist.

Um Speicherressourcen freizugeben, können zu Beginn der Ausführung von *commit_iv* die Δ-Fakten (LU_I-Δ-Fakten) gelöscht werden, die IMMEDIATE propagiert wurden und für den DEFERRED-Prozeß nicht relevant sind. Alternativ können die LU_I-Fakten auch als letzte Aktion der PL-Trigger ausgeführt werden, wodurch der Speicherplatz bereits zu einem früheren Zeitpunkt freigegeben würde. Aus Gründen einer übersichtlicheren Darstellung des Prozeßablaufs ist das Transaktionsende als Löschzeitpunkt gewählt worden.

[5] '*iv*' symbolisiert die gegenüber der COMMIT-Anweisung erweiterte Funktionalität der Integritätsprüfung ('integrity checking') und Sichtenaktualisierung ('view maintenance').

Für die relevanten Δ-Fakten wird reihenfolgeabhängig gemäß der PP-Nummer der Nettoeffekt berechnet. Die Modifikation des in der *propagation*-Relation gespeicherten Propagierungsmodus auf den Wert 'D' startet den DEFERRED-Prozeß. Zudem wird dadurch gewährleistet, daß nun die DEFERRED-PR-Trigger ausgeführt werden. Terminiert der DEFERRED-Prozeß, dann wird das Steuerungskennzeichen für den ersten IMMEDIATE-Prozeß der nächsten Transaktion wieder auf 'I' zurückgesetzt. Abschließend werden alle Δ-Basisrelationen geleert und die Transaktion mit einer COMMIT-Anweisung beendet. Die Abläufe bei der Ausführung der *commit_iv* Prozedur werden in Abbildung 5.8 veranschaulicht.

```
PROCEDURE commit_iv BEGIN
   Löschen aller Δ-Fakten aus den LU_I^Δ-Δ-Basisrelationen;
   Berechnen des Nettoeffekts der Fakten der Δ-Basisrelationen

   -- Start des DEFERRED-Prozesses
   UPDATE propagation SET mode = 'D';

   -- Zurücksetzen für den 1. IMMEDIATE-Prozeß der nächsten Transaktion
   UPDATE propagation SET mode = 'I';
   Löschen aller Δ-Fakten;
   COMMIT;
END;
```

Da DEFERRED-PR-Trigger für Einfügungen in Δ-Basisrelationen feuern, kann ein DEFERRED-Prozeß nur mittelbar durch die 'UPDATE *propagation*'-Anweisung gestartet werden. Um den DEFERRED-Prozeß zu starten, werden alle Nettoeffekt-Δ-Fakten gelöscht und wieder eingefügt, wodurch die DEFERRED-PR-Trigger feuern. Die DELETE- und INSERT-Anweisungen werden jedoch als Aktion eines $initial_D$-Triggers ausgeführt, der selbst für die Ausführung der 'UPDATE *propagation*'-Anweisung aktiviert wird. Würden die Einfügungen direkt durch die *commit_iv*-Prozedur durchgeführt, so würde jede einzelne Einfügung (unabhängige Basisfaktenänderungen) einen DEFERRED-Prozeß initiieren, statt eines gemeinsamen Prozesses wie in Folge der Ausführung des $initial_D$-Triggers. Die Löschungen und Einfügungen des $initial_D$-Triggers sind somit abhängig von der ursprünglichen Modifikationsanweisung für *propagation*, und alle gefeuerten PR-Trigger werden in die AFTER-Liste des Modifikationsereignisses eingeordnet. Dieses Detail der Ausführung eines DEFERRED-Prozesses wird ebenfalls in Abbildung 5.8 verdeutlicht.

```
CREATE TRIGGER initial_D
AFTER UPDATE ON propagation REFERENCING NEW AS neu FOR EACH ROW
WHEN T.mode = 'D'
BEGIN ATOMIC  Löschen aller Δ-Fakten;
              Wiedereinfügen aller Δ-Fakten;
END;
```

Die aufwendigen Lösch- und Einfügeaktionen ermöglichen die analoge Durchführung einer DEFERRED-Verarbeitung mit einer IMMEDIATE-Verarbeitung. Wird auf die Einfügung der Nettoeffektfakten als auslösendes Ereignis verzichtet, so muß für die DEFERRED-Propagierung ein grundlegend neues Konzept entwickelt werden, das sich in seinen wesentlichen Steuerungsmechanismen zur IMMEDIATE-Propagierung unterscheidet. Wesentlicher

Nachteil einer solchen Lösung ist, daß zentrale Triggereigenschaften, wie Transitionsvariablen und faktenspezifische Aktivierung, nicht genutzt werden können. Eine triggerbasierte Lösung wäre dann insgesamt in Frage zu stellen, und mit dem Ziel weiterer Performanzsteigerungen wäre ggf. eine prozedurale Lösung mit Δ-Sichten vorzuziehen.

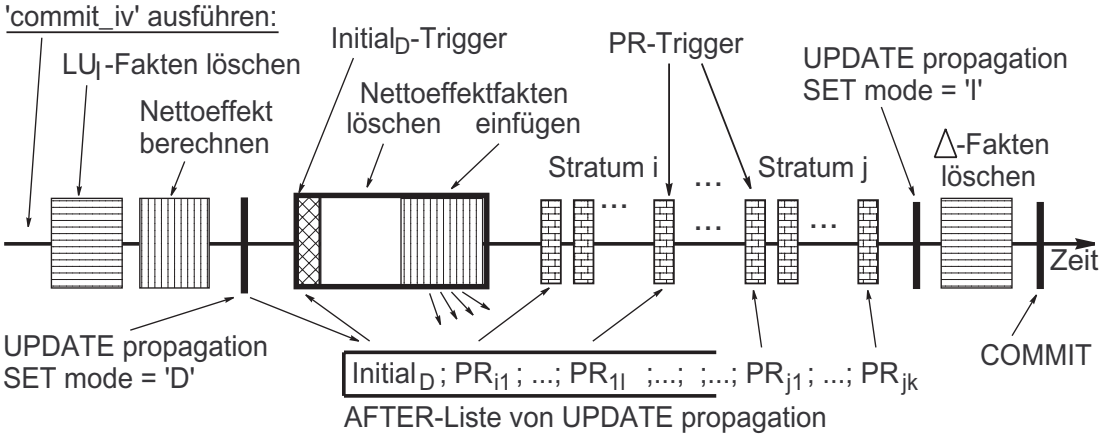

Abbildung 5.8: Triggerreihenfolge der DEFERRED-Propagierung

Präprozessor-Lösung:

Zur Lösung gehören neben der Präprozessor-Prozedur auch die '*commit_iv*'-Prozedur und der Initial-Trigger, die jedoch nur während der Transaktion ausgeführt werden.

Δ-Anfrageausdrücke und Transitionssichten:

Sei S eine SQL-Sichtdefinition der Form

$$S\ (A_1, \ldots, A_n)\ \text{AS}\ \ \text{SELECT}\ \ t_1, ..., t_n \\ \text{FROM}\ \ \ R_1\ \text{AS}\ T_{R_1},\ \ldots,\ R_m\ \text{AS}\ T_{R_m}\ \ \text{WHERE}\ \ F$$

und R_i $(1 \leq i \leq m)$ das Relationssymbol für das die Spezialisierung durchgeführt wird.

Da PR-Trigger, in denen die spezialisierten Ableitungsvorschriften angewendet werden, als ROW-Trigger definiert sind, liegen die Werte des zu propagierenden Fakts bei der Triggerausführung in den Transitionsvariablen des Triggers vor. Ein Zugriff auf die Δ-Relation ΔR_i in der FROM-Klausel der spezialisierten Ableitungsvorschrift (Δ-Anfrageausdruck) für S kann entfallen, so daß das Auftreten des Relationssymbols R_i und der Tupelvariablen T_{R_i} aus der FROM-Klausel des Δ-Anfrageausdrucks entfernt werden kann. Da das Symbol der ursprünglichen Tupelvariablen weiterhin eine gültige Tupelvariable bezeichnet, sind für diesen ersten 'Spezialisierungsschritt' keine weiteren Anpassungen der SELECT- und WHERE-Klausel erforderlich.

Die Residuenauswertung erfolgt analog zu dem Vorgehen in Datalog. Für induzierte Einfügungen werden die Residuen auf dem neuen und für Löschungen auf dem alten Zustand ausgewertet. Aufgrund der Verwendung von Tupelvariablen, ist es auch in diesem

Fall ausreichend, wenn die Relationssymbole in der FROM-Klausel ersetzt werden. Die unterschiedlichen Zustände für eine Relation R_j $(1 \leq j \leq m\ , j \neq i)$ werden mit den Exponenten ^{new}, ^{old} gekennzeichnet. Die SELECT- und WHERE-Klauseln bleiben unberührt. Diese Aussage gilt nur für induzierte Einfügungen und Löschungen. Die speziellen Anpassungen, die für die Propagierung von Modifikationen erforderlich sind, werden nach der Definition des Δ-Anfrageausdrucks erörtert.

Da nur effektive Änderungen propagiert werden sollen, werden die WHERE- Klauseln der Ableitungsvorschriften analog zu den Effektivitätstests in Datalog um Teilformeln erweitert, mittels denen für induzierte Einfügungen gegen den alten Zustand S^{old} und für induzierte Löschungen gegen den neuen Zustand S^{new} geprüft wird. Für Modifikationen wird gegen beide Zustände geprüft. Da die Δ-Relationen über das zusätzliche Attribut *pp_id* für die PP-Nummer verfügen, muß die SELECT-Liste von S um das Attribut *pp_id* erweitert werden. Die PP-Nummer des zu propagierenden Δ-Fakts wird an die *pp_id* des induzierten Δ-Fakts übergeben.

Die Transformationen einer Sichtdefinition S in Δ-Anfrageausdrücke werden ebenfalls analog zu dem Vorgehen in Datalog durchgeführt. Für jedes Relationssymbol einer persistenten Relation R_i, die in S auftritt, und jede relevante Änderungsart $(\dagger, \ddagger \in \{^+, ^-, ^\pm\})$ wird ein entsprechender Δ-Anfrageausdruck $(\Delta^\dagger S_{\Delta^\ddagger R_i}\_query)$ erzeugt. Da die Δ-Anfragen nur in PR-Triggern bereichsbeschränkte SFW-Ausdrücke sind, werden sie im Bedingungsteil von Änderungsanweisungen im Aktionsteil der PR-Trigger angewendet. Die Polarität der zu propagierenden Änderungen und die Polarität des Auftretens des Relationssymbols in S bestimmen die Polarität der induzierten Änderung.

Beispiel 5.2.1 (PR-Trigger)
Die Modellierung des PR-Triggers 'pr_1' zur Propagierung induzierter Δ^+s-Fakten aus Beispiel 5.1.2 wird den neuen Vorgaben angepaßt, indem ein ROW-Trigger definiert wird. Da die Ableitungsvorschrift für Einfügungen in die Relation 'p' spezialisiert wird, wird der Trigger 'pr_1' für Δ^+p definiert und die Tupelvariable 'tv_p' an die NEW ROW-Transitionsvariable gebunden. Da eine neue Bindung gegeben ist, wird die Zuordnung von 'tv_p' zu 'p' aus der FROM-Klausel. Die PP-Nummer des Δ^+p-Fakts wird an die 'pp_id' des Δ^+s-Fakten übergeben. Die Δ-Anfrage wird für effektive s-Fakten formuliert, so daß die Effektivität gegen 's^{old}' getestet wird. 's' sei eine FU_I-Sicht, so daß der Trigger während eines IMMEDIATE-Prozesses auszuführen ist..

TRIGGER pr_1 *AFTER INSERT ON* Δ^+p
REFERENCING NEW ROW AS tv_p *FOR EACH ROW*
WHEN (*EXISTS* (*SELECT 1 FROM propagation WHERE mode =* '*I*'))
BEGIN ATOMIC
    INSERT INTO $\Delta^+s\ (a_1,\ a_2,\ pp\_id)$ $\Delta^+s_{\Delta^+p\_query}$;
END;

$$\begin{aligned}\Delta^+s_{\Delta^+p\_query} :=\ & SELECT\ tv_p.a_1,\ tv_q.a_2,\ tv_p.pp\_id\ \ FROM\ q\ AS\ tv_q \\ & WHERE\ tv_p.a_2 = tv_q.a_1 \\ & AND\ \ NOT\ EXISTS\ (SELECT\ 1\ \ FROM\ s^{old}\ \ AS\ tv_{s^{old}} \\ & \qquad WHERE\ \ tv_p.a_1 = tv_{s^{old}}.a1 \\ & \qquad AND\ \ tv_p.a_2 = tv_{s^{old}}.a2).\end{aligned}$$

Definition 5.2.2 (Δ-Anfrageausdruck für echte Änderungen)
Sei $S \in \mathcal{R}\backslash\mathcal{RH}$ eine Sichtdefinition in einer SQL-DB $\mathcal{D}^{\mathcal{S}}$ mit einem Relationssymbol $R_i \in rel_{body}(R) \setminus rel_{help}(R)$:

$$\begin{array}{lll} S\ (A_1,\ldots,A_n)\ AS & SELECT & t_1,...,t_n \\ & FROM & R_1\ AS\ T_{R_1},\ \ldots,\ R_i\ AS\ T_{R_i},\ \ldots,\ R_m\ AS\ T_{R_m} \\ & WHERE & F \end{array}$$

T_{R_i} sei eine Tupelvariable, die an eine Transitionsvariable eines PR-Triggers $pr\_\Delta^{\dagger}S_{\Delta^{\ddagger}R_i}$ gebunden ist, und $T_{R_i}.pp\_id$ sei ein Term. Sei $*,\ \star \in \{^{new},^{old}\}$.

- *Sei $\dagger \in \{^{+},^{-}\}$, $\ddagger \in \{^{+},^{-},^{\pm}\}$. Zur Ableitung effektiver, durch $\ddagger R_i$ induzierter $\Delta^{\dagger}S$-Fakten hat der Δ-Anfrageausdruck die Form:*

$$\begin{array}{lll} \Delta^{\dagger}S_{\Delta^{\ddagger}R_i}\text{-}query\ :=\ & SELECT & t_1,...,t_n,\ T_{R_i}.pp\_id \\ & FROM & R_1^*\ AS\ T_{R_1},\ \ldots,\ R_{i-1}^*\ AS\ T_{R_{i-1}}, \\ & & R_{i+1}^*\ AS\ T_{R_{i+1}},\ \ldots,\ R_m^*\ AS\ T_{R_m} \\ & WHERE & F\ \ AND\ \ F_{effect} \end{array}$$

wobei $F_{effect} := NOT\ EXISTS\ (SELECT\ 1\ FROM\ S^{\star}\ AS\ T_{S^{\star}}\ WHERE\ T_{S^{\star}}.A_1 = t_1\ \ldots\ AND\ T_{S^{\star}}.A_n = t_n)$

und

$$\begin{array}{llll} \dagger := {}^{+}, & * := {}^{new}, & \star := {}^{old} & falls\ \ \ddagger \in \{^{+},^{\pm}\} \wedge S <_{+1} R_i \\ & & & oder\ \ \ddagger \in \{^{-},^{\pm}\} \wedge S <_{-1} R_i \\ \dagger := {}^{-}, & * := {}^{old}, & \star := {}^{new} & falls\ \ \ddagger \in \{^{+},^{\pm}\} \wedge S <_{-1} R_i \\ & & & oder\ \ \ddagger \in \{^{-},^{\pm}\} \wedge S <_{+1} R_i \end{array}$$

- *Seien F^* und F_{effect}^* die Formeln F und F_{effect}, in denen jedes Auftreten einer Tupelvariablen T_{R_j} $(1 \le j \le m \wedge j \ne i)$ eines Residuums durch $T_{R_j^*}$ ersetzt ist und jedes Auftreten eines Attributterms $T_{R_i}.A_l$ durch $T_{R_i}.A_l^*$.*
  Zur Ableitung effektiver, durch $\pm R_i$ induzierter $\Delta^{\pm}S$-Fakten hat der Δ-Anfrageausdruck die Form:

$$\begin{array}{lll} \Delta^{\dagger}S_{\Delta^{\ddagger}R_i}\text{-}query\ :=\ & SELECT & t_1^{new},...,t_n^{new},\ t_1^{old},...,t_n^{old},\ T_{R_i}.pp\_id \\ & FROM & R_1^{new}\ AS\ T_{R_1^{new}},\ \ldots,\ R_{i-1}^{new}\ AS\ T_{R_{i-1}^{new}}, \\ & & R_{i+1}^{new}\ AS\ T_{R_{i+1}^{new}},\ldots,R_m^{new}\ AS\ T_{R_m^{new}}, \\ & & R_1^{old}\ AS\ T_{R_1^{old}},\ldots,R_{i-1}^{old}\ AS\ T_{R_{i-1}^{old}}, \\ & & R_{i+1}^{old}\ AS\ T_{R_{i+1}^{old}},\ldots,R_m^{old}\ AS\ T_{R_m^{old}} \\ & WHERE & F^{new}\ \ AND\ \ F^{old} \\ & AND & F_{effect}^{old}\ \ AND\ \ F_{effect}^{new}. \end{array}$$

Um die wesentlichen Aufgaben bei einer Spezialisierung herausstellen zu können, berücksichtigt obige Definition 5.2.2 nicht alle Sonderfälle, die bei einer Ableitungsvorschrift auftreten können. Diese Fälle werden im weiteren informel erörtert.

Die Vorschriften zur Propagierung von **Modifikationen** in SQL aus Abschnitt 5.2.1 werden dem Spezialisierungsprozeß zugrunde gelegt (Definition 5.2.2). Wird eine induzierte Modifikation propagiert, so werden die neuen Werte wie eine Einfügung und die

alten wie eine Löschung propagiert. Für die alten und neuen Werte des modifizierten Fakts werden jeweils die Residuen und die Effektivitätstests über dem entsprechenden DB-Zustand ausgeführt. Da die Attributsymbole der $\Delta^{\pm}$-Relation für die Attribute der alten und neuen Werte unterschiedlich mit den Exponenten ^{new}, ^{old} gekennzeichnet sind, müssen alle Attributterme der modifizierten Relation angepaßt werden.

Induziert eine Modifikation $\pm R_i$ eine Einfügung $+S$ und Löschung $-S$, dann werden zwei Δ-Ausdrücke $(\Delta^* S_{\Delta^{\pm}R_i}\_query,\ (* \in \{^+,^-\}))$ und zwei PR-Trigger $(pr\_\Delta^* S_{\Delta^{\pm}R_i})$ generiert. Für die Modellierung der Δ-Anfrageausdrücke muß die Definition 5.2.2 hinsichlich der SELECT-Liste und der Formeln der WHERE-Klausel und des Effektivitätstests angepaßt werden. Da die Attributnamen der $\Delta^{\pm}$-Relation zur Unterscheidung mit den Exponenten ^{new}, ^{old} indiziert sind, müssen auch die Attributsymbole in den Termen des Anfrageausdrucks entsprechend gekennzeichnet werden. Ist $T_{R_i}.A$ ein beliebiger Term für ein Attribut der Relation R_i, dann wird jedes Auftreten von $T_{R_i}.A$ in den Termen der SELECT-Liste und in der Formeln F und F_{effect} der WHERE-Klausel durch $T_{R_i}.A^*$ $(* \in \{^{new}, ^{old}\})$ ersetzt. Diese Anpassung der Attributnamen ist notwendig, da bei den Projektionssichten $\Delta^{\pm+}R_i$ und $\Delta^{\pm-}R_i$ zwar auf die Indizierung der Attribute verzichtet werden kann, da die Attributsymbole nicht mehr doppelt auftreten. Da die beiden Relationen aber Sichten sind, können keine Trigger für sie definiert werden.

Ist die Spezialisierung für R_i durchzuführen, und ist R_i das einzige Relationssymbol in der FROM-Klausel, so kann das Auftreten von R_i in der FROM-Klausel z.B. durch ΔR_i mit einer neuen Tupelvariablen (**Dummy-Tupelvariable** tv_{Dummy}) ersetzt werden. Dieses Ersetzen dient allein dem Ziel, daß die Anfrage syntaktisch korrekt bleibt. In der SELECT- und WHERE-Klausel werden weiterhin die Tupelvariable verwendet, die an die Transitionsvariable des Triggers gebunden ist..

Sei Q ein für eine Relation R zu spezialisierender Anfrageausdruck mit zwei nicht rekursiven SELECT-Ausdrücken Q_1, Q_2 $(Q_1 \not\approx Q_2)$, die durch einen **UNION-Operator** verknüpft sind. Die Spezialisierung erfolgt nach Vorschriften, wie sie analog in Datalog für zwei Regeln mit dem gleichen Relationssymbol im Kopfliteral gelten. Tritt R in beiden Teilanfragen auf, so werden beide spezialisiert und angewendet. Tritt R nur in einer der Teilanfragen auf, so wird die andere nicht angewendet, da sie nicht relevant für die zu propagierenden Änderungen ist. Seien $Q_1^{\Delta}, Q_2^{\Delta}$ die für R spezialisierten Anfrageausdrücke Q_1, Q_2, dann ist Q^{Δ} ein Ausdruck der Form

$$\begin{array}{ll} Q^{\Delta} := Q_1^{\Delta} \text{ UNION } Q_2^{\Delta} & \text{gdw. } R \in Q_1 \ \wedge\ R \in Q_2 \\ Q^{\Delta} := Q_i^{\Delta} & \text{gdw. } R \in Q_i \ \wedge\ R \notin Q_j \quad \text{mit } i,j \in \{1,2\} \wedge i \neq j. \end{array}$$

Sind Q_1 ind Q_2 $(Q_1 \approx Q_2)$ zwei rekursive SELECT-Ausdrücke, dann werden auf jeden Fall (soweit wie möglich) beide Anfragen spezialisiert zu $Q^{\Delta} := Q_1^{\Delta}$ UNION Q_2^{Δ}.

Sei S eine Sichtdefinition mit dem Anfrageausdruck Q_m mit einer WITH-Klausel und den **Hilfssichten** $S_1^{help}, ..., S_{m-1}^{help}$ mit ihren Anfrageausdrücken $Q_1^{help}, ..., Q_{m-1}^{help}$. Eine für das Symbol R einer persistenten Relation $(R \in rel_{body}(S) \setminus rel_{help}(S))$ spezialisierte Ableitungsvorschrift für S wird analog zu dem Vorgehen bei dem um WITH-Klauseln erweiterten Datalog modelliert, als wären die Hilfssichten aufgefaltet (Definition 4.1.8). Unabhängig davon, ob ΔR in Q_m oder in $Q_1^{help}, ..., Q_{m-1}^{help}$ auftritt, wird die Spezialisierung auf die gleiche Art durchgeführt. Tritt ΔR in Q_j $(1 \leq j,k \leq m)$ auf, dann werden nicht nur

die Residuen von Q_j sondern auch die aller übrigen Anfragen Q_k $(k \neq j)$ über dem alten bzw. neuen Zustand ausgewertet. Da nur in Q_m die Werte des neuen Fakts zur Verfügung stehen, wird der Effektivitätstest F_{effect} ausschließlich an die Anfrage Q_m angehängt.

Wegen der zentralen Bedeutung des Begriffs der unmittelbaren (positiven/negativen) Abhängigkeiten zwischen Relationen für die Generierung spezialisierter Ausdrücke und Trigger sei an dieser Stelle auf die unterschiedliche Definition dieses Begriffs für persistente DB-Objekte und Hilfssichten hingewiesen (Definition 2.1.14). Eine persistente Relation, die indirekt von einer Sicht abhängt, wird in diesem Sinne trotzdem als unmittelbar abhängig bezeichnet, wenn die Knoten aller Abhängigkeitspfade zwischen der Relation und der Sicht Hilfssichten repräsentieren.

Aufgrund der Verwendung von AFTER-Triggern als Prozeßtrigger und des zugrunde liegenden Transaktionskonzepts liegt sowohl zum DEFERRED- wie auch zum IMMEDIATE-Propagierungsprozeß bereits die neuen Basisfakten vor. Da für dieses Propagierungsverfahren nur virtuelle Sichten zugelassen sind, kann auf die Kennzeichnung des neuen Zustands wie in Definition 5.2.2 grundsätzlich verzichtet werden, denn in allen Basisrelationen und Sichten liegt der neue Zustand bereits vor. Sei $R \in \mathcal{PS}$ ein Relationssymbol und R^{new} das Relationssymbol für die neue Faktenmenge von R, wie es in Definition 5.2.2 angewendet wird, dann gilt:

$$R^{new} \quad := \quad R.$$

Für beide Propagierungsprozesse muß somit der alte Zustand $\mathcal{F}^{old}$ mittels **Transitionssichten** simuliert werden. Der Begriff des alten Zustands bezieht sich abhängig vom Kontext der IMMEDIATE- oder DEFERRED-Propagierung auf den lokal-konsistenten Zustand vor Ausführung der aktuellen Basisfaktenänderung oder den global-konsistenten vor Ausführung der gesamten Transaktion $\mathcal{M}$. Für die Ermittlung eines lokal-konsistenten Zustands $\mathcal{F}^{old}$ müssen aus einer Menge der Δ-Fakten nur die berücksichtigt werden, die durch die aktuelle Basisfaktenänderung induziert sind.

Zur Simulation alter Zustände wird die Definition direkter und indirekter Transitionssichten (Definition 4.2.4) nach SQL übertragen und um die Möglichkeit erweitert, Modifikationen zu propagieren. Um global- wie auch lokal-konsistente Zustände ableiten zu können, wird die PP-Nummer *pp_id* der Δ-Fakten mit der aktuellen PP-Nummer (SYSTEM_PP_ID), verglichen. Während eines IMMEDIATE-Prozesses enthält die Variable SYSTEM_PP_ID die PP-Nummer der aktuellen Basisfaktenänderung und während des DEFERRED-Prozesses die der ersten Basisfaktenänderung einer Transaktion $\mathcal{M}$.

Definition 5.2.3 (Direkte und indirekte SQL-Transitionssichten)
Sei $\mathcal{D}^S$ eine NSQL-Datenbank mit der Menge der Sichtdefinitionen $\mathcal{R}$ und einer Menge von Hilfssichtdefinitionen $\mathcal{RH}$.

- *Sei $S \in \mathcal{R} \backslash \mathcal{RH}$ in $\mathcal{D}^{\mathcal{S}}$ eine Sichtdefinition der Form*

$$\begin{array}{l} \mathit{VIEW}\ \ S\ (A_1, \ldots, A_n)\ \mathit{AS} \quad [\ \mathit{WITH}\ \ S_1^{help},\ \ldots,\ S_k^{help}\ ] \\ \quad \mathit{SELECT}\ \ t_1, \ldots, t_n \\ \quad \mathit{FROM} \quad R_g\ \ \mathit{AS}\ \ T_{R_g}, \ldots, R_h\ \ \mathit{AS}\ \ T_{R_h} \\ \quad \mathit{WHERE}\ \ F, \end{array}$$

mit einer Menge aller Relationssymbole $\mathcal{R}_S := rel_{body}(S)$*, die in der Sichtdefinition auftreten, dann ist die Definition einer indirekten Transitionssicht* S^{old} *ein Ausdruck der Form*

$$\begin{array}{l} \mathit{VIEW}\ \ S^{old}\ (A_1, \ldots, A_n)\ \mathit{AS}\quad [\ \mathit{WITH}\ \ S_1^{help},\ \ldots,\ S_k^{help}\ ] \\ \quad \mathit{SELECT}\ \ t_1, \ldots, t_n \\ \quad \mathit{FROM}\quad R_g^*\ \ \mathit{AS}\ \ T_{R_g}\ , \ldots,\ R_h^*\ \ \mathit{AS}\ \ T_{R_h} \\ \quad \mathit{WHERE}\ \ F \end{array}$$

wobei $R_j^* := R_j^{old}$ *eine direkte Transitionssicht ist, gdw.* $R_j \in rel(\mathcal{B})$*,*
$R_j^* := R_j^{old}$ *eine indirekte Transitionssicht ist, gdw.* $R_j \in rel_{idb}(\mathcal{R})$*.*

- *Sei* $B \in \mathcal{B}$ *eine Basisrelation von* $\mathcal{D}^{\mathcal{S}}$ *und* $\Delta^* B$ *seien die* Δ*-Relationen für* B $(* \in \{^+, ^-, ^{\pm}, ^{\pm+}, ^{\pm-}\})$*, dann ist die Definition einer direkten Transitionssicht* B^{old} *ein Ausdruck der Form*

$$\begin{array}{l} \mathit{VIEW}\ \ B^{old}\ (A_1, \ldots, A_n)\ \mathit{AS}\ \ \mathit{WITH} \\ \quad \Delta^{plus} B\ (A_1, \ldots, A_n,\ pp\_id)\ \mathit{AS} \\ \quad\quad \mathit{SELECT}\ \ T_B.A_1, \ldots, T_B.A_n, T_B.pp\_id\ \ \mathit{FROM}\ \ \Delta^+ B\ \ \mathit{AS}\ \ T_B \\ \quad\quad \mathit{UNION} \\ \quad\quad \mathit{SELECT}\ \ T_B.A_1^{new}, \ldots, T_B.A_n^{new}, T_B.pp\_id\ \ \mathit{FROM}\ \ \Delta^{\pm} B\ \ \mathit{AS}\ \ T_B, \\ \\ \quad \Delta^{minus} B\ (A_1, \ldots, A_n,\ pp\_id)\ \mathit{AS} \\ \quad\quad \mathit{SELECT}\ \ T_B.A_1, \ldots, T_B.A_n, T_B.pp\_id\ \ \mathit{FROM}\ \ \Delta^- B\ \ \mathit{AS}\ \ T_B \\ \quad\quad \mathit{UNION} \\ \quad\quad \mathit{SELECT}\ \ T_B.A_1^{old}, \ldots, T_B.A_n^{old}, T_B.pp\_id\ \ \mathit{FROM}\ \ \Delta^{\pm} B\ \ \mathit{AS}\ \ T_B \\ \\ \quad \mathit{SELECT}\ \ T_B.A_1, \ldots, T_B.A_n\ \ \mathit{FROM}\ \ B\ \ \mathit{AS}\ \ T_B \\ \quad \mathit{WHERE}\ \ \mathit{NOT\ EXISTS}\ (\ \mathit{SELECT}\ \ \text{'1'}\ \ \mathit{FROM}\ \ \Delta^{plus} B\ \ \mathit{AS}\ \ T_{\Delta B} \\ \qquad\qquad\qquad\qquad \mathit{WHERE}\ \ T_{\Delta B}.A_1 = T_B.A_1\ \ \mathit{AND}\ \ldots \\ \qquad\qquad\qquad\qquad \mathit{AND}\quad\ \ T_{\Delta B}.A_n = T_B.A_n \\ \qquad\qquad\qquad\qquad \mathit{AND}\quad\ \ T_{\Delta B}.pp\_id \geq \mathit{SYSTEM\_PP\_ID}\ ) \\ \quad \mathit{UNION} \\ \quad \mathit{SELECT}\ \ T_{\Delta B}.A_1, \ldots, T_{\Delta B}.A_n\ \ \mathit{FROM}\ \ \Delta^{minus} B\ \ \mathit{AS}\ \ T_{\Delta B} \\ \quad \mathit{WHERE}\ \ T_{\Delta B}.pp\_id \geq \mathit{SYSTEM\_PP\_ID}. \end{array}$$

Diese Definition der Transitionssichten läßt Möglichkeiten zur performanteren Ermittlung des alten Zustands außer acht, die sich für materialisierte Sichten ergeben. In Abschnitt 5.4 wird diese Definition erweitert.

Beispiel 5.2.4 (Transitionssichten)
Für die Sichtdefinition s aus Beispiel 5.1.1 wird eine indirekte Transitionssicht s^{old} *generiert.*

$$\begin{array}{l} \mathit{VIEW}\ \ s^{old}(a_1, a_2)\ \mathit{AS}\ \ \mathit{SELECT}\ \ tv_p.a_1, tv_q.a_2\ \ \mathit{FROM}\ \ p^{old}\ \ \mathit{AS}\ \ tv_p,\ \ q^{old}\ \ \mathit{AS}\ \ tv_q \\ \qquad\qquad\qquad\qquad\qquad \mathit{WHERE}\ \ tv_p.a_2 = tv_q.a_1. \end{array}$$

Für die Basisrelation $p(a_1, a_2)$ wird eine direkte Transitionssicht p^{old} generiert.

VIEW $p^{old}\ (a_1, a_2)$ *AS WITH*
$\Delta^{plus}p\ (a_1, a_2,\ pp\_id)$ *AS*
SELECT $tv_p.a_1, tv_p.a_2, tv_p.pp\_id$ *FROM* $\Delta^{+}p$ *AS* tv_p
UNION
SELECT $tv_p.a_1^{new}, tv_p.a_2^{new}, tv_p.pp\_id$ *FROM* $\Delta^{\pm}p$ *AS* tv_p,
$\Delta^{minus}p\ (a_1, a_2, pp\_id)$ *AS*
SELECT $tv_p.a_1, tv_p.a_2, tv_p.pp\_id$ *FROM* $\Delta^{-}p$ *AS* tv_p
UNION
SELECT $tv_p.a_1^{old}, tv_p.A_2^{old}, tv_p.pp\_id$ *FROM* $\Delta^{\pm}p$ *AS* tv_p
SELECT $tv_p.a_1, tv_p.a_n$ *FROM* p *AS* tv_p
WHERE NOT EXISTS (*SELECT* '1' *FROM* $\Delta^{plus}p$ *AS* $tv_{\Delta p}$
WHERE $tv_{\Delta p}.a_1 = tv_p.a_1$ *AND* $tv_{\Delta p}.a_n = tv_p.a_n$
AND $tv_{\Delta p}.pp\_id \geq SYSTEM\_PP\_ID$)
UNION
SELECT $tv_{\Delta p}.a_1, \ldots, tv_{\Delta p}.a_n$ *FROM* $\Delta^{minus}p$ *AS* $tv_{\Delta p}$
WHERE $tv_{\Delta p}.pp\_id \geq SYSTEM\_PP\_ID.$

Präprozessor-Lösung:

Die in Abschnitt 5.2.1 vorgestellten drei Aufgaben eines Präprozessors sind im Rahmen dieses Abschnitts sukzessiv verfeinert worden, so daß abschließend folgender Algorithmus eines Präprozessors für ein Änderungspropagierungsverfahren definiert werden kann.

Algorithmus 5.2.5 (Präprozessor für ein Propagierungsverfahren)
Sei $\mathcal{S}^{\mathcal{S}} = \langle \mathcal{B}, \mathcal{RV}, \mathcal{T} \rangle$ ein DB-Schema einer SQL-DB $\mathcal{D}^{\mathcal{S}} = \langle \mathcal{F}, \mathcal{S}^{\mathcal{S}} \rangle$ mit einer Menge von Sichtdefinitionen $\mathcal{RV}$ ohne Hilfssichtdefinitionen. $\lambda_{max}(\mathcal{R})$ bezeichne das maximale Stratum von $\mathcal{R}$.

BEGIN
-- Analysieren des DB-Schemas $\mathcal{S}^{\mathcal{S}}$ der DB $\mathcal{D}^{\mathcal{S}}$;
Erstellen der Abhängigkeitsgraphen G_D und G_D^{Attr};
Klassifizieren FU-, LU-Δ-Relationen und -Integritätsbedingungen;
$\forall\ R \in \mathcal{B} \cup \mathcal{RV}$: Zuordnen der Strata $\lambda(R)$;

-- Generieren der Δ-Basisrelationen und Transitionssichten;
$\forall\ R \in \mathcal{B} \cup \mathcal{RV}$: Generieren der relevanten $\Delta^{*}R$-Basisrelationen ($* \in \{^{+}, ^{-}, ^{\pm}\}$)
$\forall\ B \in \mathcal{B}$: Generieren der relevanten direkten Transitionssicht B^{old};
$\forall\ S \in \mathcal{RV}$: Generieren der relevanten indirekten Transitionssicht S^{old};

-- Generieren der Prozeßtrigger;
$\forall\ B \in \mathcal{B}$: Generieren der relevanten PL-Trigger;
$j := 1$;
REPEAT
$\forall\ R \in \mathcal{RV} \wedge \lambda(R) = j$: Generieren der relevanten PR-Trigger;
$j := j + 1$;
UNTIL $j = \lambda_{max}(\mathcal{RV})$;
END.

Dieser Algorithmus wird in den nachfolgenden Abschnitten gemäß den verschiedenen neuen Aufgaben des Präprozessors für Verfahren zur Integritätsprüfung und Sichtenaktualisierung angepaßt und erweitert.

5.3 Inkrementelle Integritätsprüfung

Drei Themen werden in diesem Abschnitt diskutiert: das zu implementierende Integritätskonzept, die Implementierungsstrategie sowie die dafür erforderlichen Integritätsprüfungstrigger und spezialisierten Integritätsbedingungen.

Integritätskonzept:

Das in Abschnitt 3.1 erörterte NSQL-Integritätskonzept basierend auf multirelationalen, statischen ASSERTIONs, die auch auf Sichten definiert werden können, ist zu implementieren. Mit seinen beiden Prüfungszeitpunkten IMMEDIATE und DEFERRED ist das Konzept sehr gut an das SQL-Transaktionskonzept angepaßt. Wie bereits ausführlich diskutiert unterscheiden sich die NSQL-ASSERTIONs nur geringfügig von den SQL3-Integritätsbedingungen, so sind z.B. keine Aggregatfunktionen in den Formeln zugelassen. Die vordefinierten SQL3-Schlüsselbedingungen lassen sich jedoch bis auf die Möglichkeiten zur Fehlerkorrektur bei den FOREIGN KEYs und der Auszeichnung eines Schlüssels als Primärschlüssel durch ASSERTIONs simulieren. Da für ASSERTIONs beliebige NSQL-Formeln zugelassen sind, können die statischen Integritätsbedingungen sowohl als Allwie auch als Existenzaussagen formuliert werden. Zwar werden alle erzeugten ASSERTIONs unabhängig von ihrer Quantifizierung evaluiert, ein inkrementeller Ansatz wird jedoch nur für Formeln mit führendem NOT EXISTS-Operator spezifiziert. Entsprechende Transformationen, wie sie in Abschnitt 4.1.1 für existenzquantifizierte Formeln erörtert wurden, sind trivial nach SQL zu übertragen und stellen keine SQL-spezifischen Anforderungen dar.

Die Begriffe IMMEDIATE und DEFERRED werden mit der aus SQL3 bekannten Bedeutung verwendet. Anwender müssen für jede ASSERTION den Prüfungszeitpunkt definieren. Aus den beiden Prüfungszeitpunkten resultieren zwei unterschiedliche ROLLBACK-Arten. Das lokale ROLLBACK wird für einen Integritätsfehler bei der IMMEDIATE-Prüfung ausgeführt. Nur die fehlerverursachende Basisfaktenänderung und die durch sie induzierten Änderungen werden zurückgerollt. Tritt die Konsistenzverletzung bei der DEFERRED-Prüfung auf, so wird im Rahmen des globalen ROLLBACKs die gesamte Transaktion zurückgerollt.

Implementierungsstrategie:

Für die Integritätsprüfung können unmittelbar die Δ-Fakten, die das Propagierungsverfahren aus Abschnitt 5.2 bereitstellt, verwendet werden. Bei einem optimistischen Ansatz würde die Integritätsprüfung erst durchgeführt, nachdem alle Änderungen propagiert wurden. Der Implementierung wird jedoch ein pessimistischer Ansatz zugrunde gelegt. Da die Propagierung effektiver Änderungen sehr zeitaufwendig ist, wird versucht, einen Integritätsfehler so früh wie möglich zu erkennen, um unnötige Propagierungsschritte zu vermeiden. Aus diesem Grund wird die Konsistenz unmittelbar vor der Speicherung des zu prüfenden Δ-Fakts kontrolliert.

Statt wie beim generischen Verfahren spezialisierte Integritätsbedingungen als deduktive Δ-Regeln zu generieren, können hier Integritätsprüfungstrigger (IP-Trigger) generiert werden. Die hier angewendete Spezialisierung erfolgt gemäß den Vorschriften, wie sie beim generischen Verfahren definiert wurden. IP-Trigger werden als ROW-Trigger der entsprechenden Δ-Relation generiert und für die Einfügung eines Δ-Fakts ausgeführt. Im Rahmen der Ausführung wird die spezialisierte Integritätsformel angewendet und das Ergebnis interpretiert. Eine für eine Relation spezialisierte Integritätsformel und somit auch der zugehörige IP-Trigger werden dem gleichen Stratum wie die Relation zugeordnet. Da die ASSERTION dem maximalen Stratum aller direkt abhängigen Regeln zugeordnet wird, ist eine frühzeitigere Evaluierung möglich, als wenn alle spezialisierten Formeln dem gleichen Stratum wie die ursprüngliche ASSERTION angehören würden.

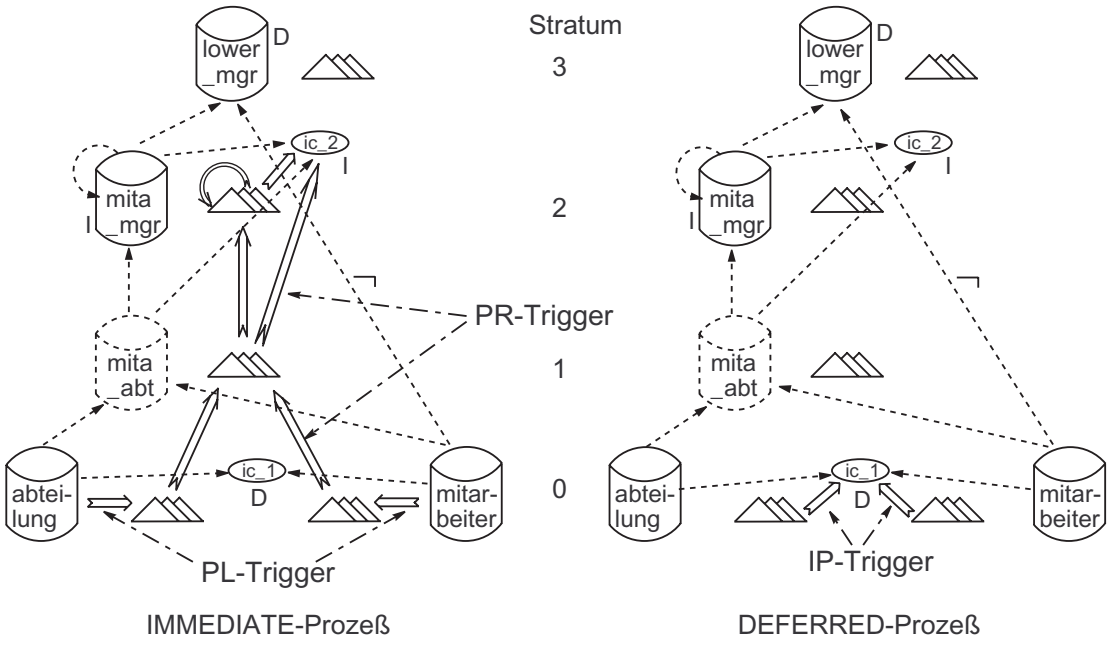

Abbildung 5.9: IMMEDIATE- und DEFERRED-Integritätsprüfung

Damit die Integritätsprüfung für ein Δ-Fakt noch vor seiner Speicherung erfolgen kann, können die IP-Trigger als **BEFORE-Trigger** generiert werden. Eine explizite Steuerung der stratumbezogenen Ausführung mittels der Triggererzeugungszeit ist für BEFORE-Trigger nicht erforderlich. Die stufenweise Ausführung des Propagierungsprozesses und die Ausführung der IP-Trigger unmittelbar vor einer Einfügeanweisung für eine Δ-Relation (Aktion von PR-Triggern), gewährleisten, daß auch die IP-Trigger stufenweise ausgeführt werden. Die Prüfung für Integritätsbedingungen auf Basisrelationen wird auch ordnungsgemäß durchgeführt, da die IP-Trigger in Folge des Protokollierungsereignisses ausgeführt werden und der PL-Trigger ein AFTER-Trigger ist, wodurch die manipulierte Basisrelation bereits im neuen Zustand vorliegt. Die Ausführungsreihenfolge der PL-, PR- und IP-Trigger wird im Detail in Abbildung 5.10 dargestellt.

Die Verwendung von BEFORE-Triggern ist jedoch nur möglich, wenn wie für das Ände-

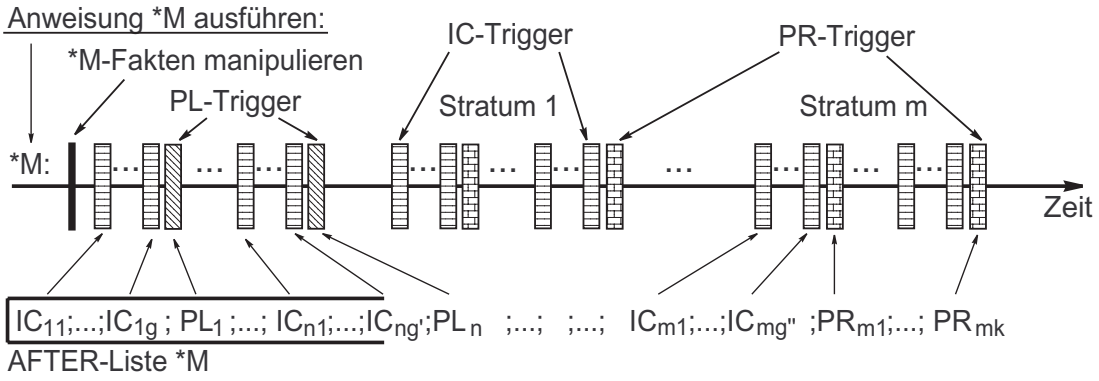

Abbildung 5.10: Triggerreihenfolge der IMMEDIATE-Integritätsprüfung

rungspropagierungsverfahren in Abschnitt 5.2 definiert, ausschließlich virtuelle Sichten zugelassen sind. Werden BEFORE-Trigger verwendet, so ist zum Zeitpunkt der Prüfung einer Integritätsbedingung für materialisierte Sichten des gleichen Stratums keine Aussage möglich, ob sie im alten Zustand, im neuen oder unvollständig vorliegen. Virtuelle Sichten hingegen werden auf den Basisrelationen angewendet, die zu diesem Zeitpunkt bereits im neuen Zustand vorliegen. Eine Realisierung auf der Basis von BEFORE-Triggern und materialisierten Sichten würde eine Ordnung der Ausführungsreihenfolge der Trigger voraussetzen und zusätzliche Transitionssichten zur Simulation des neuen Zustands erfordern. In Abschnitt 5.4 wird zur Lösung dieses Problems ein mittels AFTER-Triggern implementiertes Integritätsprüfungsverfahren erörtert.

Die Reaktion auf eine Konsistenzverletzung hängt vom Prüfungszeitpunkt ab. Bei IMMEDIATE-IP-Triggern wird nur mit dem fehlerhaften Abbruch der Ausführung des IP-Triggers reagiert. Erkennt das DB-System, daß die Ausführung eines IP-Triggers nicht ordnungsgemäß beendet wurde, wird automatisch mit der Ausführung des lokalen ROLLBACKs reagiert. Ein globales ROLLBACK für einen DEFERRED-Integritätsfehler hingegen wird in zwei Schritten durchgeführt. Zuerst wird die Ausführung des IP-Triggers fehlerhaft abgebrochen. Durch das lokale ROLLBACK terminiert der aktuelle DEFERRED-Prozeß. Anschließend wird in der *commit_iv*-Prozedur die systeminterne Variable SQLSTATE[6] abgefragt. Zeigt der Inhalt der Variablen einen Triggerabbruch an, so wird die gesamte Transaktion zurückgerollt (ROLLBACK-Anweisung), anderenfalls werden alle Änderungen der Transaktion gespeichert.

IP-Trigger und spezialisierte Integritätsformel:

Für die Realisierung der zuvor skizzierten Lösung generiert der Präprozessor für jede Integritätsbedingung $IC \in \mathcal{C}$, für jedes Auftreten einer von IC direkt abhängigen Relation R ($IC <_1 R$) ($R \in \mathcal{R} \backslash \mathcal{RH}$) und für jede relevante Änderungsart ($\ddagger \in \{^+, ^-, ^\pm\}$) einen IP-Trigger $ip\_IC_{\Delta^\ddagger R}$, dessen Aufgabe die Durchführung der inkrementellen Integritätsprüfung ist.

[6]Im ANSI-SQL-Standard ([ANSI99II] 11.38, S. 499) bleibt die Entscheidung über die Zulässigkeit von Transaktionsbefehlen in Triggern den Herstellern überlassen. Im Rahmen der vorgelegten Arbeit wird ein allgemeiner Lösungsansatz vorgestellt, der von einem Verwendungsverbot ausgeht.

Entsprechend dem Vorgehen bei der Spezialisierung der Integritätsbedingungen mit führenden Allquantoren beim generischen Verfahren wird vom Präprozessor für eine Integritätsbedingung IC eine spezialisierte Integritätsformel $IC_{\Delta^{\ddagger}R}\_condition$ generiert. Diese Integritätsformel kann als Formel in der WHEN-Klausel des IP-Triggers auftreten, so daß der Aktionsteil des Triggers nur bei einer Konsistenzverletzung ausgeführt wird. Da IP-Trigger nur bei Integritätsfehlern ausgeführt werden sollen, wird eine Allaussage (NOT EXISTS) in eine Existenzaussage (EXISTS) transformiert, die erfüllt ist, wenn ein Integritätsfehler vorliegt. Die Triggeraktion *ip_action* verursacht einen fehlerhaften Abbruch der Triggerausführung und löst somit ein lokales ROLLBACK aus.

Die WHEN-Bedingung besteht aus zwei Teilformeln. Mit der ersten wird kontrolliert, ob der aktuelle Prozeßzeitpunkt mit dem Prüfungszeitpunkt der Integritätsbedingung übereinstimmt. Die zweite Formel ist die spezialisierte Integritätsformel $IC_{\Delta^{\ddagger}R}\_condition$ ($\ddagger \in \{^{+},^{-},^{\pm}\}$). Analog zu dem Vorgehen bei den Propagierungstriggern wird auch hier auf den expliziten Zugriff auf die Δ-Relation verzichtet und auf die Transitionsvariablen zugegriffen. Die Tupelvariable T_R tritt als freie Variable in $IC_{\Delta^{\ddagger}R}\_condition$ auf und wird in der REFERENCING-Klausel an die Transitionsvariable der $\Delta^{\ddagger}R$-Relation gebunden. Entsprechend diesen Vorgaben kann ein IP-Trigger wie folgt definiert werden.

CREATE TRIGGER $ip\_IC_{\Delta^{\ddagger}R}$ BEFORE INSERT ON $\Delta^{\ddagger}R$
REFERENCING NEW ROW AS T_R FOR EACH ROW
WHEN (EXISTS (SELECT 1 FROM *propagation* WHERE *mode* = 'I' | 'D')
AND $IC_{\Delta^{\ddagger}R}\_condition$)
BEGIN ATOMIC *ip_action*; END;

Beispiel 5.3.1 (IP-Trigger)
Wird eine Spezialisierung wie beim generischen Verfahren vorausgesetzt, so kann der IP-Trigger für die Integritätsbedingung ic_1 aus Beispiel 5.1.2 zur Prüfung von Einfügungen in die p Basisrelation wie folgt modelliert werden.

TRIGGER ip_1 *BEFORE INSERT ON* $\Delta^{+}p$
REFERENCING NEW ROW AS tv_p *FOR EACH ROW*
WHEN (*EXISTS* (*SELECT 1 FROM propagation WHERE mode = 'D'*)
AND EXISTS (*SELECT* $tv_p.a_1, tv_p.a_2$
FROM $\Delta^{+}p$ *AS* tv_{dummy} *WHERE* $tv_p.a_1 > tv_p.a_2$))
BEGIN ATOMIC ip_action; END.

Da das Relationssymbol $\Delta^{+}p$ nur eingefügt ist, um die syntaktische Korrektheit der Anfrage zu gewährleisten, kann die Integritätsbedingung wie folgt vereinfacht werden:

TRIGGER ip_1 *BEFORE INSERT ON* $\Delta^{+}p$
REFERENCING NEW ROW AS tv_p *FOR EACH ROW*
WHEN (*EXISTS* (*SELECT 1 FROM propagation WHERE mode = 'D'*)
AND $tv_p.a_1 > tv_p.a_2$)
BEGIN ATOMIC ip_action; END.

Die Formeln von **Existenzaussagen** werden nicht spezialisiert sondern nur negiert, damit sie erfüllt sind, wenn ein Integritätsfehler vorliegt. Für jede existenzquantifizierte

Integritätsbedingung IC und jede von IC direkt positiv/negativ abhängige Relation R ($IC <_{+1} R$ / $IC <_{-1} R$) werden IP-Trigger generiert, die für Löschungen/Einfügungen bzw. die alten/neuen Werte eines modifizierten Fakts aktiviert werden.

Gemäß der Idee einer attributbezogenen Relevanzanalyse wird die WHEN-Klausel eines IP-Triggers zur Propagierung von Modifikationen um eine dritte Teilformel erweitert, die sicherstellt, daß die Evaluierung nur bei Wertänderungen der für IC relevanten Attribute $(A_g, ..., A_h)$ durchgeführt wird:[7]

AND ($T_R.A_g^{new} \neq T_R.A_g^{old}$ OR ... OR $T_R.A_h^{new} \neq T_R.A_h^{old}$).

Definition 5.3.2 (Spezialisierte Integritätsformel)
Sei eine ASSERTION $IC \in \mathcal{C}$

ASSERTION IC *CHECK* (*NOT EXISTS* ([*WITH* [*RECURSIVE*] $R_1^{help}, ..., R_m^{help}$]
SELECT $t_1, ..., t_n$
FROM R_g *AS* $T_{R_g}, ..., R_i$ *AS* $T_{R_i}, ..., R_h$ *AS* T_{R_h}
WHERE (F)))
IMMEDIATE | *DEFERRED*,

einer SQL-DB $\mathcal{D}^{\mathcal{S}}$ *gegeben, dann ist die spezialisierte Integritätsformel* $IC_{\Delta^{\ddagger}R_i}\_condition$ ($\ddagger \in \{^+, ^-, ^{\pm}\}$) *ein Ausdruck der Form*

$IC_{\Delta^{\ddagger}R_i}\_condition$:= *EXISTS* ([*WITH* [*RECURSIVE*] $R_1^{help\_new}, ..., R_m^{help\_new}$]
SELECT $t_1, ..., t_n$
FROM R_g^{new} *AS* $T_{R_g}, ..., R_{i-1}^{new}$ *AS* $T_{R_{i-1}}$,
R_{i+1}^{new} *AS* $T_{R_{i+1}}, ..., R_h^{new}$ *AS* T_{R_h}
WHERE (F))

Sei $\ddagger := {}^{\pm}$ *und* A_l *ein Attribut der Relation* R_i ($attr(A_l) \in attr(R_i)$).
Ist $IC <_{-} R_i$ *bzw.* $IC <_{+} R_i$ *dann wird jedes Auftreten eines Terms* $T_{R_i}.A_l$ *in* $IC_{\Delta^{\ddagger}R_i}\_condition$ *durch* $T_{R_i}.A_l^{new}$ *bzw.* $T_{R_i}.A_l^{old}$ *ersetzt.*

Tritt R als einziges Relationssymbol in einer FROM-Klausel von IC auf, so wird er durch den Bezeichner der Δ-Basisrelation $\Delta^{\ddagger}R$ ersetzt[8].

Für die Durchführung des zweiten Schritts der Simulation eines globalen ROLLBACKs wird die Prozedur *commit_iv* um die Abfrage der systeminternen Variablen SQLSTATE erweitert, die den Abbruch einer Triggerausführung anzeigt. Nach der Anweisung zur Speicherung des Kennzeichens 'D' wird die Prozedur wie folgt modifiziert.

```
Löschen aller Δ-Fakten;
IF  SQLSTATE = 'Triggerabbruch'
  THEN  ROLLBACK;
  ELSE  UPDATE propagation SET mode = 'I';  COMMIT;
END IF;
```

[7] Für die Verwendung des logischen OR-Operators im NSQL-Kontext gelten analoge Aussagen wie bei der Definition der PR-Trigger (Abschnitt 5.2.2).

[8] Wie ein solcher, nur mit dem Ziel syntaktischer Korrektheit formulierter Ausdruck, optimiert werden kann, wurde bereits zuvor bei der Definition der Δ-Anfragen erläutert.

Wird der im Datalog-Kontext formulierte Relevanzbegriff auf die Syntax von SQL-ASSERTIONs IC mit führendem NOT EXISTS-Operator angewendet (Allaussagen), so ist eine Änderung $\ddagger R$ für eine Relation R, die in IC auftritt, relevant, gdw.

$$(\ddagger \in \{^+, {}^\pm\} \ \wedge \ IC <_- R) \quad \vee \quad (\ddagger \in \{^-, {}^\pm\} \ \wedge \ IC <_+ R).$$

Präprozessor-Lösung:

Der Algorithmus 5.2.5 eines Präprozessors für ein Propagierungsverfahren muß für ein inkrementelles Integritätsprüfungsverfahren nur um die Generierung der IP-Trigger erweitert werden. Da die IP-Trigger als BEFORE-Trigger definiert sind, ist der Zeitpunkt der Generierung ohne Bedeutung, und sie können somit nach den PL- und noch vor den PR-Triggern generiert werden.

BEGIN
  −− Analysieren des DB-Schemas $\mathcal{S}^{\mathcal{S}}$ der DB $\mathcal{D}^{\mathcal{S}}$;
  ...
  $\forall\ IC \in \mathcal{C}$: Zuordnen der Strata $\lambda(IC)$;

  −− Generieren der Δ-Basisrelationen und Transitionssichten;
  −− Generieren der Prozeßtrigger;
  $\forall\ B \in \mathcal{B}$: Generieren der relevanten PL-Trigger;
  $\forall\ IC \in \mathcal{C}$: Generieren der relevanten IP-Trigger;
  $j := 1$;
  REPEAT
    $\forall\ R \in \mathcal{RV} \wedge \lambda(R) = j$: Generieren der relevanten PR-Trigger;
    $j := j + 1$;
  UNTIL $j = \lambda_{max}(\mathcal{RV})$;
END.

Da bei keinem der derzeitigen kommerziellen DB-Systeme ASSERTIONs zulässige DB-Objekte sind, und auch die ASSERTION-Formeln in diesen DB-Systemen keine zulässigen Bedingungsausdrücke sind, muß die Verwaltung der ASSERTIONs vollständig vom Präprozessor auf der Basis externer Schemarelationen geleistet werden. Zur Schemaevolution werden Präprozessorprozeduren (z.B. *create_assertion*, *alter_assertion*) aufgerufen, denen als Parameter die ASSERTION-Definitionen übergeben werden. Als erste Aktion wenden diese Prozeduren die ursprüngliche Formel der ASSERTION an, um die Konsistenz der Faktenbasis festzustellen. Liegt eine Konsistenzverletzung vor, so wird die Schemamanipulation entsprechend den Vorgaben des SQL-Standards zurückgewiesen. Ist die Konsistenz für die neue Integritätsbedingung gegeben, so werden weitere Präprozessorprozeduren für die Generierung der für eine inkrementelle Evaluierung erforderlichen DB-Objekte aufgerufen.

5.4 Inkrementelle Sichtenaktualisierung

Da vom ANSI-Standardisierungsgremium nur ein Konzept für virtuelle Sichten spezifiziert worden ist und die Implementierungen materialisierter Sichten in kommerziellen DB-Systemen funktional beschränkt sind, wird zu Beginn dieses Abschnitts ein Konzept für virtuelle und materialisierte Sichten in SQL definiert. Anschließend wird eine

Implementierungsstrategie entwickelt, deren Grundlage das in Abschnitt 5.2 definierte Propagierungsverfahren ist.

Konzept materialisierter und virtueller Sichten:

Das Konzept materialisierter Sichten wird für Anwendungsfälle mit sehr unterschiedlichen Anforderungen entwickelt. Um den Vorteil des effizienteren Zugriffs auf die Fakten materialisierter Sichten auch während einer Transaktion nutzen zu können und dabei aufgrund des SQL-Transaktionskonzepts bereits auf den neuen Zustand zugreifen zu können, ist es nicht ausreichend, nur zum Transaktionsende zu aktualisieren. Es muß eine IMMEDIATE-Aktualisierung unmittelbar nach der Ausführung einer Basisfaktenänderung möglich sein. Für Data Warehouse-Anwendungen und verteilte Datenbanken ist es hingegen ausreichend, die Sichten zum Transaktionsende zu aktualisieren (DEFERRED-Aktualisierung). In Abschnitt 4.3.6 sind Verfahren vorgestellt worden, die die beiden Aktualisierungszeitpunkte nach dem Transaktionsende ON DEMAND und periodisch berücksichtigen. Die dort vorgeschlagenen Verfahren ([SBLC00], [BDDF+98] und [CGLMT96], [CKLMR97]) lassen sich mit nur geringfügigen Erweiterungen in das hier entwickelte Verfahren integrieren. Da diese Anforderungen unabhängig vom SQL-Kontext sind, werden sie nicht weiter diskutiert.

In einem DB-Schema können sowohl virtuelle als auch materialisierte Sichten definiert werden. Zwischen den Sichten dieser beiden Typen können beliebige Abhängigkeitsverhältnisse (hierarchisch oder rekursiv) bestehen. Die Menge aller Sichten entspricht einer stratifizierbaren Datalog-Regelmenge. Das SQL-Konzept **virtueller Sichten** bleibt unberührt von der Erweiterung um materialisierte Sichten. Lediglich die Semantik virtueller Sichten, die von materialisierten abhängen, muß definiert werden. Bislang ist die Semantik virtueller Sichten durch ihre Anwendung auf die Fakten der Basisrelationen definiert worden. Hängt eine virtuelle Sicht von einer materialisierten Sicht ab, so stellt sich die Frage, ob die Ableitungsvorschrift auf den materialisierten Fakten der Sicht (IMMEDIATE: neuer Zustand; DEFERRRED: alter Zustand) oder auf den Basisfakten (neuer Zustand) angewendet wird. Um die Effizienzvorteile materialisierter Sichten zu jedem Zeitpunkt nutzen zu können, wird vorausgesetzt, daß virtuelle Sichten, die sich auf materialisierte Sichten beziehen, zu jedem Zeitpunkt auf deren Faktenmenge angewendet werden.

Der SQL-Befehl zur Erzeugung einer virtuellen Sicht kann intuitiv um die zur Erzeugung einer materialisierten Sicht erforderlichen Optionen erweitert werden. Sei S ein Relationssymbol, A_1, ..., A_n Attributsymbole und Q eine SQL-Anfrage, dann ist eine Ausdruck der Form

CREATE IMMEDIATE | DEFERRED MATERIALIZED | VIRTUAL
[RECURSIVE] VIEW S ($A_1, \ldots, A_n$) AS Q.

eine DDL-Anweisung eines erweiterten SQLs zur Erzeugung einer Sicht.

Implementierungsstrategie:

Die Materialisierung abgeleiteter Fakten kann mittels Basisrelationen simuliert werden, indem der Präprozessor für jede Definition einer materialisierten Sicht $S \in \mathcal{RM}$ eine Basisrelation $S \in \mathcal{B}$ mit der gleichen Attributstruktur generiert. Damit für den Anwender

die Simulation transparent ist, werden für die materialisierte Sicht und die Basisrelation das gleiche Relationssymbol verwendet. Das nicht eindeutige Relationssymbol ist unproblematisch, da Basisrelationen im internen Schema des DB-Systems verwaltet werden und materialisierte Sichten im externen Schema des Präprozessors. Zur Unterscheidung zwischen beiden Objektarten wird im Rahmen dieser Arbeit die materialisierte Sicht als S^{mat} dargestellt. Anwender erhalten nur Zugriffsrechte für S. S^{mat} ist kein existierendes DB-Objekt, ihre Ableitungsvorschrift wird nur in den Schemarelationen des Präprozessors gespeichert. Zum Zeitpunkt der Erzeugung von S^{mat} wird ihr Anfrageausdruck Q angewendet, um für die erste Materialisierung die S^{mat}-Fakten in S zu speichern.

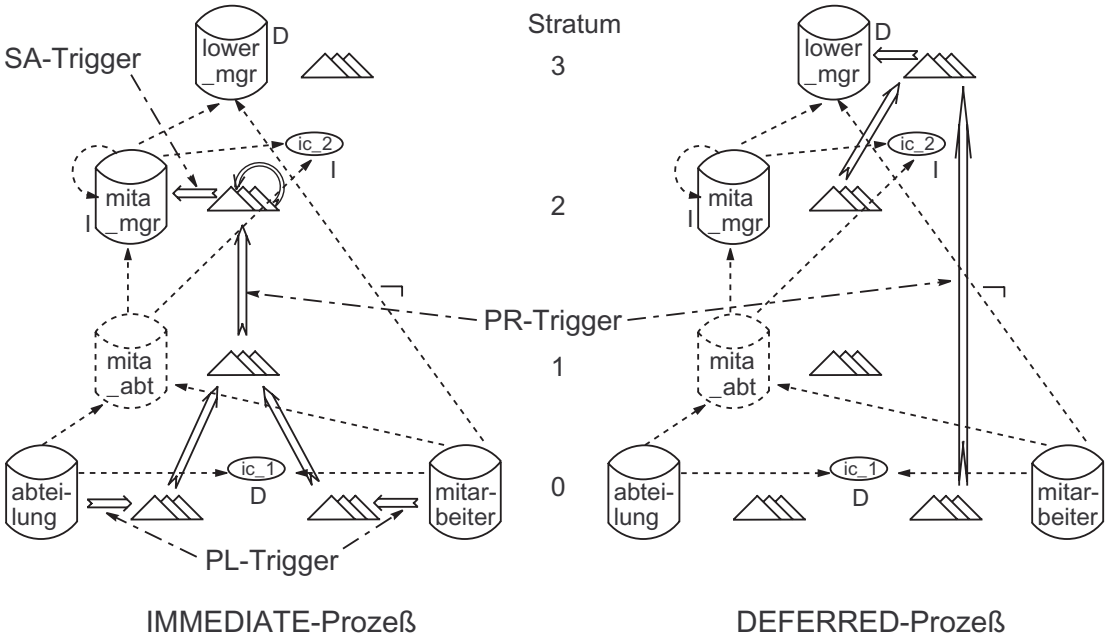

Abbildung 5.11: IMMEDIATE- und DEFERRED-Sichtenaktualisierung

Der Implementierungsvorschlag für ein Verfahren zur inkrementellen Sichtenaktualisierung wird, wie in Abbildung 5.11 skizziert, als Erweiterung des in Abschnitt 5.2 definierten Propagierungsverfahrens entwickelt. Die in der Basisrelation S gespeicherten Fakten werden mittels Sichtenaktualisierungstriggern (SA-Triggern) inkrementell aktualisiert, die als AFTER ROW-Trigger definiert werden können. Da die für die Aktualisierung relevanten Informationen in den Δ-Relationen vorliegen, kann für jede Δ-Basisrelation einer materialisierten Sicht ein SA-Trigger generiert werden, der bei der Einfügung von Δ-Fakten feuert. Abhängig davon, ob der SA-Trigger für eine $\Delta^+/\Delta^-/\Delta^\pm$-Relation definiert ist, wird eine Einfügung/Löschung/ Modifikation auf S zum definierten Aktualisierungszeitpunkt IMMEDIATE oder DEFERRED ausgeführt.

Diese Aktualisierung kann als Phase im Anschluß an den Propagierungsprozeß durchgeführt werden mit der Konsequenz, daß alle materialisierten Sichten während des Propagierungsprozesses im alten Zustand vorliegen. Alternativ können die materialisierten Sichten während des Prozesses zum Ende der Ableitung der Δ-Fakten eines Stratums aktualisiert werden, so daß bei der Ableitung der Δ-Fakten höherer Sichten die mate-

rialisierten Sichten niedrigerer Strata bereits im neuen Zustand vorliegen. In Hinblick auf die etwaige Integration einer Integritätsprüfung, kann es von Vorteil sein, wenn zum Prüfungszeitpunkt die materialisierten Sichten bereits im neuen Zustand vorliegen, damit der Aufwand für eine Simulation mittels Transitionssichten entfällt. Angepaßt an die Reihenfolgesteuerung der PR-Trigger sind die Erzeugungszeiten der SA-Trigger des Stratums i kleiner als die des Stratums $i+1$. Damit die SA-Trigger eines Stratums i aber erst ausgeführt werden, wenn alle Δ-Fakten des Stratums vorliegen, ist ihre Erzeugungszeit größer als die der PR-Trigger des Stratums i. Die Abfolge der PR- und SA-Trigger bei der Ausführung zweier Strata i und j wird in Abbildung 5.12 skizziert. Sie ist unabhängig vom Prozeßzeitpunkt.

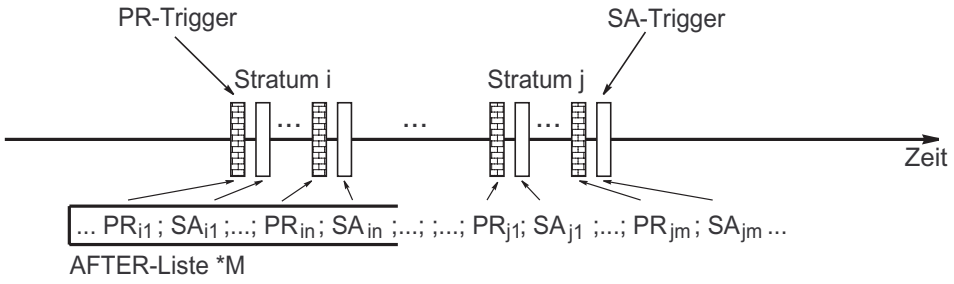

Abbildung 5.12: Triggerreihenfolge der Sichtenaktualisierung

Soll auch im Rahmen des Propagierungsprozesses der Vorteil der Materialisierung der abgeleiteten Fakten genutzt werden, so gestaltet sich das Konzept der Transitionssichten komplexer als für den Änderungspropagierungsprozeß in Abschnitt 5.2.2. Für Basisrelationen werden weiterhin die direkten Transitionssichten generiert und für virtuelle Sichten weiterhin die indirekten Transitionssichten für den alten Zustand (Definition 5.2.3). Für materialisierte Sichten werden aber zwei Transitionssichten generiert. Ihre Anwendung hängt davon ab, welchem Stratum die Sicht angehört, von der sie referenziert wird.

- Gehört die referenzierende Sicht R einem höheren Stratum an als S^{mat} ($\lambda(R) > \lambda(S^{mat})$), dann liegt die Basisrelation S im neuen Zustand vor, und die Δ-Fakten liegen auch vollständig vor, so daß der alte Zustand S^{old} mittels direkter Transitionssichten simuliert werden kann. Da für die Basisrelation zur Simulation der virtuellen Sicht das gleiche Relationssymbol wie für die materialisierte Sicht verwendet wird, sind keine Anpassungen der Ableitungsvorschriften von Sichten höherer Strata erforderlich.

- Sind die referenzierende Sicht R und S^{mat} rekursiv abhängig ($\lambda(R) \approx \lambda(S^{mat})$), dann liegt während des Ableitungsprozesses die Basisrelation S im alten Zustand vor. In welchem Zustand die Δ-Relationen vorliegen ist unbekannt, so daß der neue Zustand S^{new} mittels indirekter Transitionssichten simuliert wird. Für die Residuenauswertung rekursiv abhängiger Sichten reicht es aus, S^{new} nur für materialisierte Sichten zu generieren, die rekursiv abhängig sind. Da jedoch nur effektive Änderungen propagiert werden, wird aufgrund der Effektivitätstests für jede materialisierte Sicht die indirekte Transitionssicht S^{new} generiert.

Erweiterungen des Änderungspropagierungsverfahrens:

Für die Ausführung der ermittelten induzierten Änderungen auf materialisierten Sichten können vom Präprozessor folgende **SA-Trigger** generiert werden. Sei S^{mat} eine materialisierte Sicht ($S^{mat} \in \mathcal{RM}$) mit n Attributen $A_1, ..., A_n$. Sei S die zugehörige Basisrelation und $\Delta^{\dagger}S$ die relevanten Δ-Basisrelationen ($\dagger \in \{^{+},^{-},^{\pm}\}$). Für jedes $\Delta^{\dagger}S$ wird ein SA-Trigger $sa\_\Delta^{\dagger}S$ generiert.

CREATE TRIGGER $sa\_\Delta^{\dagger}S$
AFTER INSERT ON $\Delta^{\dagger}S$
REFERENCING NEW ROW AS T_S FOR EACH ROW
WHEN (EXISTS (SELECT 1 FROM *propagation* WHERE *mode* = 'I' | 'D'))
BEGIN ATOMIC *sa_action*; END;

wenn	dann	
$\dagger = ^{+}$	*sa_action* :=	INSERT INTO S VALUES ($T_S.A_1, ..., T_S.A_n,$);
$\dagger = ^{-}$	*sa_action* :=	DELETE FROM S
		WHERE $A_1 = T_S.A_1$ AND...AND $A_n = T_S.A_n$;
$\dagger = ^{\pm}$	*sa_action* :=	UPDATE S
		SET $A_1 = T_S.A_1^{new}, ..., A_n = T_S.A_n^{new}$,
		WHERE $A_1 = T_S.A_1^{old}$ AND...AND $A_n = T_S.A_n^{old}$.

Beispiel 5.4.1 (SA-Trigger)
Für die Sicht s aus Beispiel 5.1.2 kann folgender SA-Trigger modelliert werden.

TRIGGER sa_1 *AFTER INSERT ON* $\Delta^{+}s$
FOR EACH ROW REFERENCING NEW ROW AS tv_s
BEGIN ATOMIC INSERT INTO $s\ (a_1, a_2)$
VALUES ($tv_s.a_1,\ tv_s.a_2$); *END.*

Die Definition 5.2.3 der **direkten und indirekten Transitionssichten** muß erweitert werden, so daß direkte Transitionssichten zur Simulation des alten Zustands für Basisrelationen und materialisierte Sichten erzeugt werden und indirekte Transitionssichten zur Simulation des alten Zustands virtueller Sichten. Für die Simulation des neuen Zustands materialisierter Sichten wird für den Fall, daß rekursiv abhängige Sichten auf S^{new} zugreifen oder im Rahmen von Effektivitätstests auf S^{new} zugegriffen wird, eine weitere indirekte Transitionssicht definiert.

Definition 5.4.2 (Indirekte Transitionssicht S^{new})
Sei $S \in \mathcal{RM}$ die Definition einer materialisierten Sicht einer SQL-Datenbank $\mathcal{D}^{\mathcal{S}}$ und $\{R_1, ..., R_n\} = rel_{body}(\mathcal{R}) \setminus rel_{help}(\mathcal{R})$ die Menge der Relationssymbole direkt von S abhängiger Relationen ($S <_1 R_i$) ($1 \leq g, h, i \leq n$), dann ist eine indirekte Transitionssicht zur Simulation des neuen Zustands S^{new} ein Ausdruck der Form

VIEW $S^{new}\ (A_1, \ldots, A_n)$ *AS* [*WITH* $S_1^{help}, ..., S_k^{help}$]
SELECT $t_1, ..., t_n$ *FROM* R_g^{*} *AS* $T_{R_g}, \ldots,$ R_h^{*} *AS* T_{R_h} *WHERE* F

wobei $* \in \{^{new},^{old}\}$ *und*
$R_i^{*} := R_i^{new}$ *gdw.* $S \approx R_i \ \wedge \ R_i \notin \{S_1^{help}, ..., S_k^{help}\}$
$R_i^{*} := R_i$ *gdw.* $\neg(S \approx R_i) \ \vee \ R_i \in \{S_1^{help}, ..., S_k^{help}\}$.

Bei der Verwendung der verschiedenen Transitionssichten von materialisierten Sichten müssen der Zeitpunkt des Zugriffs, die Art der Abhängigkeit und das Stratum des referenzierenden DB-Objekts beachtet werden. Sei S^{mat} eine materialisierte Sicht mit der Basisrelation S, die von einem DB-Objekt $R \in \mathcal{R} \cup \mathcal{C}$ referenziert wird.

- S liegt im neuen Zustand vor, gdw. $R < S \ \wedge \ \neg(R \approx S)$.
- S liegt im alten Zustand vor, gdw. $R \approx S$.
- S^{old} simuliert den alten Zustand, gdw. $\neg(R \approx S)$.
- S^{new} simuliert den neuen Zustand, gdw. $R \approx S$.

Aus diesen Zusammenhängen resultieren die nachfolgenden Anpassungen der bisherigen Definitionen aus dem Änderungspropagierungsverfahren in Abschnitt 5.2.2. Bei der Definition 5.2.2 der Δ-Anfragen für echte Änderungen müssen bei den Residuen und den Effektivitätstests die materialisierten Sichten explizit berücksichtigt werden. Sei $S \in rel_{head}(\mathcal{R})$ das Relationssymbol einer Sicht aus einer DB $\mathcal{D}^{\mathcal{D}}$ und $R \in rel(\mathcal{R})$ das Symbol einer Relation, das als Residuum in der spezialisierten Ableitungsvorschrift von S auftritt. $S^{new}, S^{old}, R^{new}, R^{old}$ bezeichnen die Transitionssichten für die neuen und alten Zustände der Relationen. $\mathcal{RM} \subseteq \mathcal{R}$ sei eine Menge von Sichtdefinitionen materialisierter Sichten.

- Ist $S \in \mathcal{RM} \ \wedge$ wird F_{effect} gegen den
  neuen Zustand S^{new} ausgewertet, dann gilt: $S^{new} := S^{new}$,
  alten Zustand S^{old} ausgewertet, dann gilt: $S^{old} := S$.
- Ist $R \in \mathcal{RM} \ \wedge \ \mathcal{R} \approx \mathcal{S} \ \wedge$ wird das Residuum R gegen den
  neuen Zustand R^{new} ausgewertet, dann gilt: $R^{new} := R^{new}$,
  alten Zustand R^{old} ausgewertet, dann gilt: $R^{old} := R$.
- Ist $R \in \mathcal{RM} \ \wedge \ \neg(\mathcal{R} \approx \mathcal{S}) \ \wedge$ wird das Residuum R gegen den
  neuen Zustand R^{new} ausgewertet, dann gilt: $R^{new} := R$,
  alten Zustand R^{old} ausgewertet, dann gilt: $R^{old} := R^{old}$.

Die Definition 5.2.3 der indirekten Transitionssichten zur Simulation des alten Zustands S^{old} einer Sicht S muß für den Fall angepaßt werden, daß Relationssymbole materialisierter Sichten in der Ableitungsvorschrift der Sichtdefinition auftreten. Seien $\mathcal{B}, \mathcal{RV}, \mathcal{RM}, \mathcal{RH}$ die Mengen der Basisrelationen, der virtuellen, materialisierten Sichten und Hilfssichten eines DB-Schemas $\mathcal{S}^{\mathcal{S}}$, und sei R das Symbol einer Relation, das in der Ableitungsvorschrift von S auftritt ($S >_1 R$), dann wird der alte Zustand von R abhängig vom Relationstyp wie folgt ermittelt:

R^{old} ist eine direkte Transitionssicht,	gdw. $R \in rel(\mathcal{B})$,
R^{old} ist eine indirekte Transitionssicht,	gdw. $R \in rel(\mathcal{RV} \backslash \mathcal{RH})$,
$R^{old} := R$	gdw. $R \in rel(\mathcal{RH})$,
$R^{old} := R$	gdw. $R \in rel(\mathcal{RM}) \ \wedge \ R \approx S$,
$R^{old} := R^{old}$	gdw. $R \in rel(\mathcal{RM}) \ \wedge \ \neg(R \approx S)$.

Präprozessor-Lösung:

Da für das zugrunde liegende DB-Schema $\mathcal{S}^{\mathcal{S}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{T} \rangle$ die Restriktion einer SQL-Datenbank, daß die Menge der Sichtdefinitionen materialisierter Sichten leer ist ($\mathcal{RM} = \emptyset$), aufgehoben ist, ergeben sich drei Erweiterungen des Algorithmus 5.2.5 für einen Präprozessors eines Propagierungsverfahrens. Ein Präprozessor für ein Sichtenaktualisierungsverfahren generiert die zur Simulation der materialisierten Sichten erforderlichen Basisrelationen und Transitionssichten zur Ableitung alter wie neuer Zustände ebenso wie die SA-Trigger zur Durchführung der inkrementellen Aktualisierung.

BEGIN
  −− Analysieren des DB-Schemas $\mathcal{S}^{\mathcal{S}}$ der DB $\mathcal{D}^{\mathcal{S}}$;
  −− Generieren der Δ-Basisrelationen und Transitionssichten;
  $\forall\ R \in \mathcal{B} \cup \mathcal{R}$: Generieren der relevanten $\Delta^* R$-Basisrelationen ($* \in \{^+, ^-, ^\pm\}$)
  $\forall\ B \in \mathcal{B}$: Generieren der relevanten direkten Transitionssicht B^{old};
  $\forall\ S \in \mathcal{R}$: Generieren der relevanten indirekten Transitionssicht S^{old};
  $\forall\ S \in \mathcal{RM}$: Generieren der relevanten indirekten Transitionssicht S^{new};
  $\forall\ S \in \mathcal{RM}$: Generieren der Basisrelationen S zur Simulation von S^{mat};

  −− Generieren der Prozeßtrigger;
  $\forall\ B \in \mathcal{B}$: Generieren der relevanten PL-Trigger;
  $j := 1$;
  REPEAT
    $\forall\ R \in \mathcal{R} \quad \wedge\ \lambda(R) = j$: Generieren der relevanten PR-Trigger;
    $\forall\ R \in \mathcal{RM} \wedge\ \lambda(R) = j$: Generieren der relevanten SA-Trigger;
    $j := j + 1$;
  UNTIL $j = \lambda_{max}(\mathcal{RV} \cup \mathcal{RM})$;
END.

Da materialisierte Sichten keine zulässigen SQL3-DB-Objekte sind, und da das hier definierte Konzept von den derzeitigen kommerziellen DB-Systemen nicht bzw. nicht vollständig unterstützt wird, muß ihre Verwaltung vollständig vom Präprozessor auf der Basis externer Schemarelationen gewährleistet werden. Zur Schemaevolution werden Präprozessor-Prozeduren (z.B. *create_materialized_view*, *alter_materialized_view*) aufgerufen, denen als Parameter die Sichtdefinitionen übergeben werden. Als erste Aktion generieren diese Prozeduren die für die Simulation erforderlichen Basisrelationen und speichern alle für die Sichten ableitbaren Fakten in den Basisrelationen (erste Materialisierung). Des weiteren werden Präprozessorprozeduren für die Generierung der für eine inkrementelle Aktualisierung erforderlichen DB-Objekte aufgerufen.

5.5 Inkrementelles Verfahren zur simultanen Integritätsprüfung und Sichtenaktualisierung

In den beiden vorangegangenen Abschnitten sind unabhängig voneinander Verfahren zur inkrementellen Integritätsprüfung und Sichtenaktualisierung in SQL entwickelt worden. Unter Ausnutzung der Optimierungsmöglichkeiten für den gegebenen Kontext ist bei beiden Verfahren versucht worden, die Laufzeit während der Transaktion möglichst zu minimieren. Bei dem abschließend entwickelten Verfahren für beide Aufgabenstellungen sind

nun Restriktionen und Optimierungsmöglichkeiten, die aus dem jeweils anderen Verfahren resultieren, zu beachten bzw. zu nutzen.

Das in Abschnitt 5.3 definierte Integritätskonzept und das in Abschnitt 5.4 definierte Konzept materialisierter Sichten mit inkrementeller Sichtenaktualisierung sind in einem gemeinsamen Verfahren zu realisieren. Als Integritätsbedingungen sind NSQL-ASSERTIONs mit den beiden Integritätsprüfungszeitpunkten IMMEDIATE und DEFERRED zugelassen. Materialisierte Sichten sind genauso ausdrucksstark wie virtuelle und können zu den beiden Zeitpunkten IMMEDIATE und DEFERRED aktualisiert werden. Beide Sichttypen können in einem Schema definiert werden, und ASSERTIONs sind auf Sichten definierbar.

Unmittelbar nach der Ausführung einer Basisfaktenänderung werden die IMMEDIATE-ASSERTIONs evaluiert und die IMMEDIATE-Sichten aktualisiert. Δ-Fakten werden nur soweit propagiert, wie sie im IMMEDIATE-Prozeß verarbeitet werden (FU_I-Δ-Fakten). Erst auf dem neuen lokal-konsistenten Zwischenzustand kann ein Anwender Anweisungen ausführen. Zum Ende der Transaktion werden die DEFERRED–ASSERTIONs evaluiert und die DEFERRED-Sichten aktualisiert. Zu diesem Zweck wird der reihenfolgeabhängige Nettoeffekt der Δ-Fakten der gesamten Transaktion ermittelt. Nur für diese Nettoeffekt-Δ-Fakten werden die Δ-Fakten propagiert, die ausschließlich im DEFERRED-Prozeß verarbeitet werden (FU_D-Δ-Fakten).

Damit für materialisierte Sichten und sie referenzierende Integritätsbedingungen eine sinnvolle Semantik gewährleistet ist, werden zwei Restriktionen formuliert. DEFERRED-Sichten können ausschließlich von DEFERRED-Sichten und Integritätsbedingungen referenziert werden. Für IMMEDIATE-Sichten gelten keine Einschränkungen hinsichtlich der von ihnen abhängigen Sichten und Integritätsbedingungen.

Implementierungsstrategie und Anpassungen der Verfahren:

Für die Implementierung der materialisierten Sichten kann das in Abschnitt 5.4 entwickelte Verfahren zur Änderungspropagierung und Sichtenaktualisierung unverändert zugrunde gelegt werden. Das Verfahren aus Abschnitt 5.3 zur inkrementellen Integritätsprüfung basiert jedoch auf der Annahme, daß ausschließlich virtuelle Sichten vorliegen. Um im Rahmen der Integritätsprüfung die Vorteile der Materialisierung abgeleiteter Fakten nutzen zu können, wird das Integritätsprüfungsverfahren modifiziert.

Die Prüfung der auf einer materialisierten Sicht S^{mat} definierten ASSERTIONs mittels BEFORE-Triggern kann zu längeren Laufzeiten während der Transaktion führen, da zum Zeitpunkt der BEFORE-Triggerausführung der Zustand von S noch unbekannt ist, so daß S^{new} nur mittels indirekter Transitionssichten simuliert werden kann. Um ohne zusätzlichen Ableitungsaufwand auf die aktualisierten Fakten zugreifen zu können, reicht es aus, die Integritätsprüfung für die induzierten Änderungen eines Stratums zum Ende der Bearbeitung der Sichten eines Stratums auszuführen, nachdem die SA-Trigger alle materialisierten Sichten des Stratums aktualisiert haben. Dies kann erreicht werden, indem die **IP-Trigger als AFTER-Trigger** generiert werden mit einer Erzeugungszeit, die größer ist als die der SA-Trigger des gleichen Stratums. Nachteilig bei einer Konsistenzverletzung ist die unnötige Aktualisierung, bevor der Fehler zum Ende der Bearbeitung

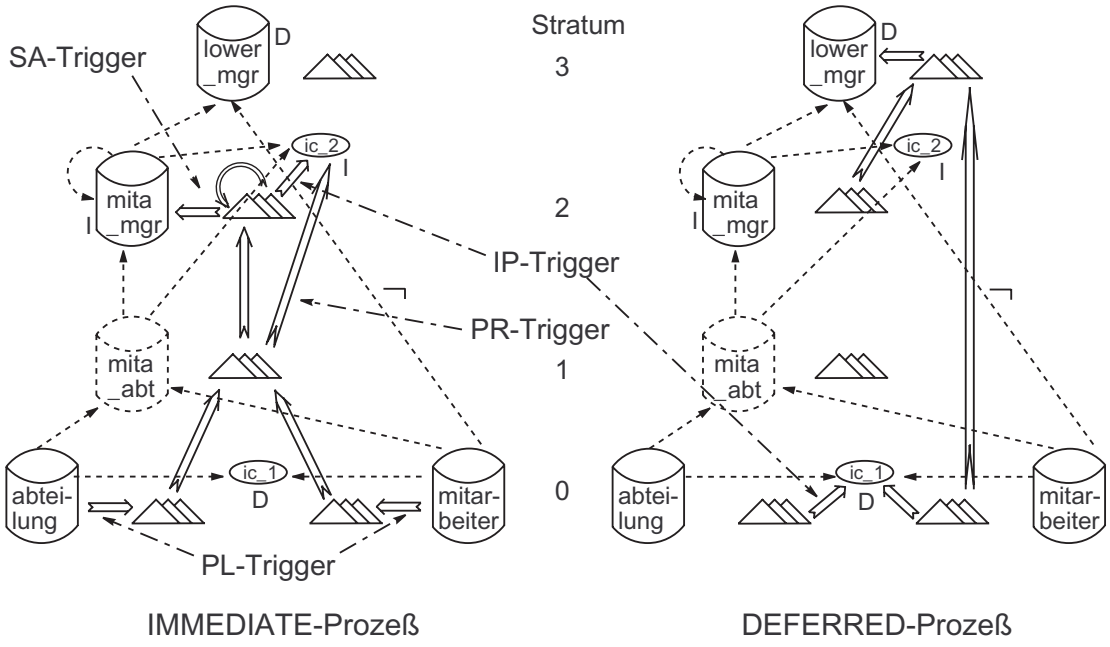

Abbildung 5.13: Integritätsprüfung und Sichtenaktualisierung

des Stratums erkannt werden kann. Da die Residuen in den Integritätsbedingungen immer auf dem neuen Zustand angewendet werden, ist es jedoch von Vorteil, daß der neue Zustand bei der Integritätsprüfung bereits in den Relationen vorliegt und nicht simuliert werden muß.

Zudem muß bei der Verwendung materialisierter Sichten die in Abschnitt 5.3 vorgenommene **Stratazuordnung der IP-Trigger** neu definiert werden. Sind nur virtuelle Sichten zugelassen, so ist der Aufwand bei der Ableitung unabhängig davon, zu welchem Zeitpunkt (Stratum) sie durchgeführt wird. In Abschnitt 5.3 wird mit dem Ziel einer frühestmöglichen Evaluierung eine ASSERTION bei der Bearbeitung des Stratums $\lambda(R)$ geprüft, in dem die Δ-Fakten der relevanten Relation R propagiert werden. Hängt eine ASSERTION direkt von materialisierten Sichten ab, die einem höheren Stratum als $\lambda(R)$ oder dem gleichen Stratum zugeordnet sind, so liegen diese Sichten entweder im alten oder in einem unbekannten Zustand vor. In beiden Fällen wäre der neue Zustand durch Anwendung der R^{new}-Transitionssichten zu simulieren.

Die neue Stratazuordnung wird nicht für eine ASSERTION IC durchgeführt, sondern für die spezialisierten Integritätsformeln $IC_{\Delta^{\ddagger}R}$, so daß die verschiedenen IP-Trigger einer ASSERTION je nach relevanter Relation R verschiedenen Strata zugeordnet sein können. Ist IC direkt oder indirekt abhängig von materialisierten Sichten, so werden die spezialisierten Integritätsformeln frühestens angewendet, wenn alle abhängigen materialisierten Sichten im neuen Zustand vorliegen. Anderenfalls wird die bisherige Vorschrift zur Stratazuordnung beibehalten. $\lambda(R_k)$ sei das maximale Stratum aller von IC direkt oder indirekt abhängigen materialisierten Sichten.

$$\lambda(IC_{\Delta^{\ddagger}R}) := \lambda(R_k) \qquad \text{gdw.} \quad \lambda(R) < \lambda(R_k)$$
$$\lambda(IC_{\Delta^{\ddagger}R}) := \lambda(R) \qquad \text{sonst.}$$

Bei dieser Stratazuordnung steht dem Synergieeffekt der einmaligen Propagierung und mehrfachen Verwendung der Nachteil gegenüber, daß im Falle einer Konsistenzverletzung ggf. unnötige Propagierungs- und Verarbeitungsschritte durchgeführt werden, weil für ein im Stratum i ermitteltes Δ-Fakt der Integritätsfehler erst im Stratum $i+x$ $(x > 0)$ erkannt wird. Welcher der beiden Prüfungszeitpunkte, Stratum i oder $i + x$, im Fehlerfall performanter ist, ist sehr anwendungsspezifisch und kann nur empirisch entschieden werden. Die Entscheidung, die Integritätsprüfung erst nach der Sichtenaktualisierung durchzuführen, widerspricht zwar der Intention des SQL-Standards, unnötige Aktionen im Fehlerfall zu vermeiden, sie stellt jedoch im Fall der Konsistenz einen beachtlichen Effizienzgewinn dar, da weniger Simulationen durchzuführen sind.

Triggererzeugungszeit		Triggererzeugungszeit
PR-, SA-, IP-Trigger des Stratums i	$<$	PR-, SA-, IP-Trigger des Stratums $i+1$
PL-Trigger (Stratum 0)	$<$	IP-Trigger des Stratums 0
IP-Trigger des Stratums 0	$<$	PR-Trigger des Stratums 1
PR-Trigger des Stratums i $(i > 0)$	$<$	SA-Trigger des Stratums i $(i > 0)$
SA-Trigger des Stratums i $(i > 0)$	$<$	IP-Trigger des Stratums i $(i > 0)$
generierte Prozeßtrigger	$<$	benutzerdefinierte Trigger

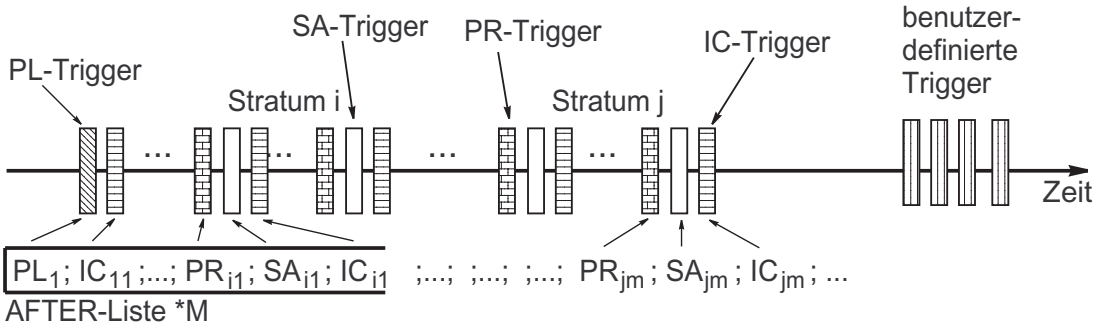

Abbildung 5.14: Trigger der Integritätsprüfung und Sichtenaktualisierung

In der Übersicht werden die Abhängigkeiten zwischen den Triggererzeugungszeiten der verschiedenen Triggerarten zusammengefaßt. Werden diese Vorschriften vom Präprozessor berücksichtigt, dann ist die Ausführung eines iterativen Fixpunktprozesses gewährleistet. In Abbildung 5.14 wird die zeitliche Abfolge der Triggerausführungen skizziert.

Die Erweiterungen, die in den letzten drei Abschnitten für den Algorithmus 5.2.5, durchgeführt wurden, werden in diesem Algorithmus eines Präprozessors für eine simultan ausgeführte Integritätsprüfung und Sichtenaktualisierung zusammengefaßt.

Algorithmus 5.5.1 (Präprozessor)
Sei $\mathcal{S}^{\mathcal{S}} = \langle \mathcal{B}, \mathcal{R}, \mathcal{C}, \mathcal{T} \rangle$ *ein DB-Schema einer SQL-DB* $\mathcal{D}^{\mathcal{S}} = \langle \mathcal{F}, \mathcal{S}^{\mathcal{S}} \rangle$ *mit einer Menge virtueller und materialisierter Sichten ohne Hilfssichten* ($\mathcal{R}:= \mathcal{RV} \cup \mathcal{RM}$). $\mathcal{C}_\Delta$ *sei die Menge der spezialisierten Integritätsformeln* IC_Δ *mit* $IC \in \mathcal{C}$. $\lambda_{max}(\mathcal{R})$ *bezeichne das maximale Stratum einer Sichtenmenge.*

BEGIN

-- Analysieren des DB-Schemas $\mathcal{S}^{\mathcal{S}}$ der DB $\mathcal{D}^{\mathcal{S}}$;
Erstellen der Abhängigkeitsgraphen G_D und G_D^{Attr};
Klassifizieren FU-, LU-Δ-Relationen und -Integritätsbedingungen;
$\forall\ R \in \mathcal{R} \cup \mathcal{B}$: *Zuordnen der Strata $\lambda(R)$;*

-- Generieren der Δ-Basisrelationen und Transitionssichten;
$\forall\ R \in \mathcal{B} \cup \mathcal{R}$: *Generieren der relevanten $\Delta^* R$-Basisrelationen* $(* \in \{^+, ^-, ^\pm\})$
$\forall\ B \in \mathcal{B}$: *Generieren der relevanten direkten Transitionssicht B^{old};*
$\forall\ S \in \mathcal{R}$: *Generieren der relevanten indirekten Transitionssicht S^{old};*
$\forall\ S \in \mathcal{RM}$: *Generieren der relevanten indirekten Transitionssicht S^{new};*
$\forall\ S \in \mathcal{RM}$: *Generieren der Basisrelation S zur Simulation von S^{mat};*

-- Generieren der Prozeßtrigger;
$\forall\ B \in \mathcal{B}$: *Generieren der relevanten PL-Trigger;*
$\forall\ IC \in \mathcal{C}$: *Generieren der relevanten IC_Δ-Integritätsformeln;*
$\forall\ IC_\Delta \in \mathcal{C}_\Delta$: *Zuordnen der Strata $\lambda(IC_\Delta)$;*
$j := 0$;
REPEAT
  $\forall\ R \in \mathcal{R} \wedge \lambda(R) = j$: *Generieren der relevanten PR-Trigger;*
  $\forall\ R \in \mathcal{RM} \wedge \lambda(R) = j$: *Generieren der relevanten SA-Trigger;*
  $\forall\ IC_\Delta \in \mathcal{C}_\Delta \wedge \lambda(C) = j$: *Generieren der relevanten IP-Trigger;*
  $j := j + 1$;
UNTIL $j = \lambda_{max}(\mathcal{RV} \cup \mathcal{RM} \cup \mathcal{C})$;

END.

5.6 Anwendungsbeispiel

Für das aus Kapitel 2 bekannte Beispiel 3.1.4 einer Unternehmensorganisation werden in diesem Abschnitt exemplarisch einige Prozeßtrigger definiert. Ausgehend von diesen Beispieltriggern[9] sind die noch fehlenden Trigger intuitiv definierbar.

Aufgrund der Abhängigkeits- und Relevanzanalyse werden die Δ-Basisrelationen vom Präprozessor den entsprechenden $\mathcal{FU}^\Delta$-/$\mathcal{LU}^\Delta$-Klassen zugeordnet.

$\mathcal{FU}_I^\Delta := \{\Delta^* mitarbeiter, \Delta^* abteilung, \Delta^* mita\_abt, \Delta^* mita\_mgr\}$;
$\mathcal{FU}_D^\Delta := \{\Delta^* lower\_mgr\}$;
$\mathcal{LU}_I^\Delta := \{\Delta^+ abteilung, \Delta^* mita\_abt\}$.

Anhand dieser Informationen ermittelt der Präprozessor den Zeitpunkt der Ausführung der PR-Trigger und den Zeitpunkt der frühestmöglichen Löschung, der IMMEDIATE-propagierten Fakten. Der Zeitpunkt der Ausführung der IP- und SA-Trigger wird unmittelbar anhand des Prüfungs-/Aktualisierungszeitpunkts der ASSERTIONs und materialisierten Sicht ermittelt werden.

[9] Die Indizierung der verschiedenen Bezeichner im NSQL-Kontext stellt zwar keinen gültigen NSQL-Bezeichner dar, sie wird hier aber verwendet, um die Analogie zu den in den Definitionen verwendeten Bezeichnern aufrechtzuerhalten.

Protokolltrigger:

Als ein Ergebnis der Relevanzanalyse wird für jede Relation (Basisrelation, Sicht) und jede Änderungsart ($* \in \{^+, ^-, ^\pm\}$) eine Δ-Basisrelation generiert. Für jede der Δ-Relationen der Basisrelationen *mitarbeiter* und *abteilung* generiert der Präprozessor einen Protokolltrigger *protokoll_*Δ^**mitarbeiter*, *protokoll_*Δ^**abteilung*.

IP-Trigger:

Für die Bedingung ic_1 werden vier DEFERRED-IP-Trigger generiert, für Einfügungen/Modifikationen in *mitarbeiter* und für Löschungen/Modifikationen in *abteilung*. Da beide referenzierten Relationen dem Stratum 0 angehören, werden die IP-Trigger ebenfalls dem Stratum 0 zugeordnet. ic_1 wird DEFERRED geprüft. Tritt das Relationssymbol, für das die Spezialisierung durchgeführt werden soll, als einziges Relationssymbol in der FROM-Klausel auf, so kann die Formel wesentlich vereinfacht werden. Der führende SELECT-Ausdruck entfällt und die NOT EXISTS-Quantifizierung resultiert daraus, daß mit der Formel der Fehlerfall formuliert wird, auf den dann im Aktionsteil mit dem Abbruch der Triggerausführung reagiert wird. Der Prüfungszeitpunkt ist DEFERRED ('D').

CREATE TRIGGER $ip\_ic_{1\Delta^+ mitarbeiter}$ AFTER INSERT ON $\Delta^+ mitarbeiter$
REFERENCING NEW ROW AS tv_m FOR EACH ROW
WHEN (EXISTS (SELECT 1 FROM *propagation* WHERE *mode* = 'D')
AND NOT EXISTS (SELECT 1 FROM *abteilung* AS tv_a
WHERE $tv_a.nr = tv_m.abteil$))
BEGIN ATOMIC *ip_action*; END;

Bei Modifikationen wird die WHEN-Bedingung aufgrund der Ergebnisse der attributbezogenen Relevanzanalyse um eine Teilformel erweitert, so daß die restliche Formel und der Triggeraktionsteil nur bei Wertänderungen relevanter Attribute angewendet werden.

CREATE TRIGGER $ip\_ic_{1\Delta^\pm mitarbeiter}$ AFTER INSERT ON $\Delta^\pm mitarbeiter$
REFERENCING NEW ROW AS tv_m FOR EACH ROW
WHEN (EXISTS (SELECT 1 FROM *propagation* WHERE *mode* = 'D')
AND $(tv_a.nr^{old} \neq tv_a.nr^{new})$
AND NOT EXISTS (SELECT 1 FROM *abteilung* AS tv_a
WHERE $tv_a.nr^{new} = tv_m.abteil$))
BEGIN ATOMIC *ip_action*; END;

Für die Integritätsbedingung ic_2 werden, obwohl die *mitarbeiter*-Relation zweimal auftritt, nur vier IMMEDIATE-IP-Trigger generiert, je einer für Einfügungen/Modifikationen bezüglich *mita_mgr* und für Löschungen/Modifikationen bezüglich *mita_abt*. Bei der UNION-Verknüpfung werden beide SFW-Ausdrücke, in denen das Relationssymbol *mita_abt* auftritt, in einem Ausdruck spezialisiert, analog zu der Vorgehensweise bei zwei Datalog-Regeln mit dem gleichen Rumpfliteral. Hinsichtlich der Modifikationen von *mita_mgr* ist nur das Attribut *mgr* relevant und für *mita_abt* nur die beiden Attribute *bez* und *mita*. Da die Sicht *mita_mgr* von *mita_abt* abhängt, müssen alle IP-Trigger dem Stratum 2 zugeordnet werden.

SA-Trigger:

SA-Trigger werden nur für die materialisierten Sichten *mita_mgr* und *lower_mgr* generiert. Da für jede der Sichten alle drei Änderungsarten relevant sind, werden jeweils drei sa_Δ^*-Trigger generiert ($* \in \{^+,^-,^\pm\}$), die die Fakten der Basisrelationen *mita_mgr*, *lower_mgr* aktualisieren.

CREATE TRIGGER $sa\_\Delta^\pm mita\_mgr$ AFTER INSERT ON $\Delta^\pm mita\_abt$
REFERENCING NEW ROW AS tv_{mm} FOR EACH ROW
WHEN (EXISTS (SELECT 1 FROM *propagation* WHERE *mode* = 'I'))
BEGIN ATOMIC
  UPDATE *mita_mgr*
  SET $mita = tv_{mm}.mita^{new}$, $bez = tv_{mm}.bez^{new}$, $mgr = tv_{mm}.mgr^{new}$
  WHERE $mita = tv_{mm}.mita^{old}$, $bez = tv_{mm}.bez^{old}$, $mgr = tv_{mm}.mgr^{old}$;
END;

PR-Trigger:

- Für die Sicht *mita_abt*:
  Da beide Relationen *mitarbeiter* und *abteilung* positiv auftreten, tritt kein Polaritätswechsel zwischen induzierender und induzierter Änderung auf. Die PR-Trigger $pr\_\Delta^* mita\_abt_{\Delta^*\dagger}$ ($* \in \{^+,^-,^\pm\}$, $\dagger \in \{mitarbeiter, abteilung\}$) werden generiert. Modifikationen werden als solche nur propagiert, wenn ausschließlich bei den Attributen *mitarbeiter.mita*, *gehalt*, *mgr* und *abteilung.bez* Werte verändert werden. Modifikationen der Attribute *mitarbeiter.abteil* oder *abteilung.nr* werden als induzierte Einfügungen/Lösungen propagiert. Dadurch, daß es von *mita_abt* abhängige Relationen gibt, die IMMEDIATE zu verarbeiten sind, werden IMMEDIATE-PR-Trigger modelliert.

- Für die Sicht *mita_mgr*:
  Für jedes Auftreten einer Relation, die keine Hilfssicht ist, in einem Anfrageausdruck oder in einer der Hilfssichtspezifikationen wird ein PR-Trigger generiert (nur *mita_abt*). Da die SELECT-Ausdrücke, in denen das Relationssymbol von *mita_abt* auftritt, durch einen UNION-Operator verknüpft werden, werden beide Auftreten des Relationssymbols in einem PR-Trigger spezialisiert. Aufgrund des Aktualisierungszeitpunkts wird ein IMMEDIATE-PR-Trigger generiert.

- Für die Sicht *lower_mgr*:
  Für die referenzierte Relation *mitarbeiter* muß der Polaritätswechsel beachtet werden. Modifikationen der *mita_mgr*-Relation werden als solche nur propagiert, wenn ausschließlich bei den Attributen *mita_mgr.bez* und *mgr* Werte verändert werden. Für die Modifikation des Attributs *mita_mgr.mita* werden zwei PR-Trigger zur Ableitung der induzierten Δ^+- und Δ^--*lower_mgr*-Fakten generiert. Da die *mitarbeiter*-Relation in einer Unteranfrage auftritt, werden Modifikationen nur als induzierte Einfügungen/Löschungen propagiert. *lower_mgr* ist als FU_D-Relation klassifiziert, so daß DEFERRED-PR-Trigger generiert werden.

Die Sicht *mita_abt* wird für die Einfügungen in die *mitarbeiter*-Basisrelation spezialisiert. Das Seitenliteral *abteilung* wird über dem neuen Zustand ausgewertet. Für den Effektivitätstest wird die direkte Transitionssicht $mita\_abt^{old}$ angewendet.

```
CREATE TRIGGER pr_Δ+mita_abt_Δ+mitarbeiter
AFTER INSERT ON Δ+mitarbeiter
REFERENCING NEW ROW AS tv_m FOR EACH ROW
WHEN (EXISTS (SELECT 1 FROM propagation WHERE mode = 'I' ) )
BEGIN ATOMIC
   INSERT INTO Δ+mita_abt
   SELECT t_m.mita, t_m.abteil, t_a.bez, t_m.mgr FROM abteilung AS t_a
   WHERE t_m.abteil = t_a.nr
   AND   NOT EXISTS (
         SELECT 1 FROM mita_abt^old AS tv_ma^old
         WHERE tv_ma^old.mita = tv_m.mita AND tv_ma^old.nr = tv_m.abteil
         AND   tv_ma^old.bez = tv_a.bez   AND tv_ma^old.mgr = tv_m.mgr);
END;
```

mita_mgr ist eine rekursive Sicht, deren Ableitungsvorschrift für Löschungen aus der Sicht *mita_abt* spezialisiert wird. *mita_abt* tritt zweimal in der rekursiven Hilfssicht auf. Da die beiden SELECT-Ausdrücke durch einen UNION-Operator verknüpft sind, werden beide Auftreten der Relationssymbole in einem Trigger spezialisiert. Da das Relationssymbol in der Verankerung der Rekursion als einziges Symbol in der FROM-Klausel auftritt, wird es durch ein 'Dummy'-Symbol und eine 'Dummy'-Tupelvariable ersetzt, die nicht weiter verwendet werden. Da h_1 eine Hilfsregel ist, wird für das Seitenliteral h_1^{old} keine Transitionssicht angewendet. Für den Effektivitätstest wird auf die indirekte Transitionssicht $mita\_mgr^{new}$ zugegriffen.

```
CREATE TRIGGER pr_Δ-mita_mgr_Δ-mita_abt
AFTER INSERT ON Δ-mita_abt
REFERENCING NEW ROW AS tv_ma FOR EACH ROW
WHEN (EXISTS (SELECT 1 FROM propagation WHERE mode = 'I' ) )
BEGIN ATOMIC
   INSERT INTO Δ-mita_mgr WITH RECURSIVE
     h_1^old(mita, bez, mgr) AS
         SELECT tv_h1.mita, tv_h1.bez, tv_ma.mgr FROM h_1^old AS tv_h1
         WHERE tv_h1.mgr = tv_ma.mita
         UNION
         SELECT tv_ma.mita, tv_ma.bez, tv_ma.mgr FROM Δ-mita_abt AS tv_Dummy

     SELECT tv_h1.mita, tv_h1.bez, tv_h1.mgr FROM h_1^old AS tv_h1
     WHERE NOT EXISTS (
           SELECT 1 FROM mita_mgr^new AS tv_mm^new
           WHERE tv_mm^new.mita = tv_h1.mita AND tv_mm^new.bez = tv_h1.bez
           AND   tv_mm^new.mgr = tv_h1.mgr);
END;
```

Transitionssichten:

Für die beiden Basisrelationen wird jeweils eine direkte Transitionssicht für den alten Zustand generiert, für die virtuelle Sicht *mita_abt* eine indirekte Transitionssicht für den alten Zustand, für die materialisierten Sichten jeweils eine zur Simulation des alten (direkte) und eine für den neuen Zustand (indirekte Transitionssicht).

CREATE VIEW $mita\_abt^{old}$ $(mita, nr, bez, mgr)$ AS
SELECT $t_m.mita, t_m.abteil, t_a.bez, t_m.mgr$
FROM $mitarbeiter^{old}$ AS t_m, $abteilung^{old}$ AS t_a
WHERE $t_m.abteil = t_a.nr$;

CREATE VIEW $mita\_mgr^{new}$ $(mita, bez, mgr)$ AS WITH RECURSIVE
$h_1(mita, bez, mgr)$ AS SELECT $tv_{h_1}.mita, tv_{h_1}.bez, tv_{ma}.mgr$
FROM h_1 AS tv_{h_1}, $mita\_abt$ AS tv_{ma}
WHERE $tv_{h_1}.mgr = tv_{ma}.mita$
UNION
SELECT $tv_{ma}.mita, tv_{ma}.bez, tv_{ma}.mgr$
FROM $mita\_abt$ AS tv_{ma}
SELECT $tv_{h_1}.mita, tv_{h_1}.bez, tv_{h_1}.mgr$ FROM h_1 AS tv_{h_1}.

Ausführung einer Transaktion:

Obwohl in den Trigger-Ausführungslisten alle gefeuerten Trigger (IMMEDIATE- und DEFERRED-Prozeßtrigger) eingeordnet werden, werden wegen einer besseren Lesbarkeit in diesem Beispiel jeweils nur die für den aktuellen Zeitpunkt relevanten Prozeßtrigger aufgelistet. Während des IMMEDIATE-Prozesses wird somit auf die Auflistung der DEFERRED-Trigger verzichtet, die aufgrund ihrer WHEN-Bedingung nur während des DEFERRED-Prozesses ausgeführt werden. Folgende Basisfakten und abgeleitete Fakten liegen vor:

mitarbeiter

Hugo	10	5000	Anton
Erna	20	5000	Helga
Emil	10	7000	Helga
Helga	13	9000	

abteilung

10	EK	Einkauf
20	VK	Verkauf
13	MG	Management

mita_abt (virt.)

Hugo	10	EK	Anton
Erna	20	VK	Helga
Emil	10	EK	Helga
Helga	13	MG	

mita_mgr (mat.)

Hugo	EK	Anton
Erna	VK	Helga
Emil	EK	Helga
Helga	MG	

lower_mgr (mat.)

Hugo	EK	Anton

Es wird die Transaktion $\mathcal{M}$ ausgeführt:

$\mathcal{M}$:= { $+mitarbeiter(Anton, 10, 6000, Emil)$; commit_iv; }

mitarbeiter

Hugo	10	5000	Anton
Erna	20	5000	Helga
Emil	10	7000	Helga
Helga	13	9000	
Anton	10	6000	Emil

mita_abt (virt.)

Hugo	10	EK	Anton
Erna	20	VK	Helga
Anton	10	EK	Emil
Emil	10	EK	Helga
Helga	13	MG	

Die Einfügung $+mitarbeiter(Anton, 10, 6000, Emil)$ feuert den PL-Trigger und startet somit unmittelbar einen **IMMEDIATE-Prozeß**.

Triggerabarbeitungsliste des Ereignisses $+mitarbeiter(Anton, 10, 6000, Emil)$:

$\text{AFTER}_{mitarbeiter} := \{ (pl\_\Delta^{+}mitarbeiter,\ +mitarbeiter(\text{Anton, 10, 6000, Emil})) \}$

Die Ausführung des PL-Trigger-Ereignispaares gewährleistet, daß in $\Delta^{+}mitarbeiter$ das neue Fakt eingefügt wird. Dieses Ereignis feuert unabhängig vom Verarbeitungsmodus den IP-Trigger für ic_1 sowie die PR-Trigger für *mita_abt*, *lower_mgr*. Da aber aufgrund des Prozeßzeitpunkts nur $pr\_\Delta^{+}mita\_abt_{\Delta^{+}mitarbeiter}$ relevant ist, wird das Trigger-Ereignispaar als einziges hier im Beispiel aufgeführt.

Δ^{+}mitarbeiter

Anton 10 6000 Emil

$\text{AFTER}_{mitarbeiter} :=$
$\{ (pr\_\Delta^{+}mita\_abt_{\Delta^{+}mitarbeiter},\ +\Delta^{+}mitarbeiter(\text{Anton, 10, 6000, Emil})) \}$

Die Ausführung des $pr\_\Delta^{+}mita\_abt_{\Delta^{+}mitarbeiter}$-Trigger-Ereignispaares führt zur Speicherung von $\Delta^{+}mita\_abt$-Fakten, wodurch der PR-Trigger $pr\_\Delta^{+}mita\_mgr_{\Delta^{+}mita\_abt}$ und der IP-Trigger $ip\_ic_{\Delta^{+}mita\_abt}$ gefeuert werden.

Δ^{+}mita_abt

Anton 10 EK Emil

$\text{AFTER}_{mitarbeiter} :=$
$\{ (pr\_\Delta^{+}mita\_mgr_{\Delta^{+}mita\_abt},\ +\Delta^{+}mita\_abt(\text{Anton, 10, EK, Emil}));$
$(ip\_ic_{\Delta^{+}mita\_abt},\ +\Delta^{+}mita\_abt(\text{Anton, 10, EK, Emil})) \}$

Die Propagierung der induzierten *mita_mgr*-Einfügungen erfolgt mittels Ausführung der Trigger-Ereignispaare $pr\_\Delta^{+}mita\_mgr_{\Delta^{+}mita\_abt}$, woraufhin für jedes Δ-Fakt ein SA- und ein IP-Trigger gefeuert werden.

Δ^{+}mita_mgr

Anton	EK	Emil	Hugo	EK	Emil
Anton	EK	Helga	Hugo	EK	Helga

$\text{AFTER}_{mitarbeiter} :=$
$\{ (sa\_\Delta^{+}mita\_mgr,\ +\Delta^{+}mita\_mgr(\text{Anton, 10, EK, Emil}));$
$(sa\_\Delta^{+}mita\_mgr,\ +\Delta^{+}mita\_mgr(\text{Anton, EK, Helga}));$
$(sa\_\Delta^{+}mita\_mgr,\ +\Delta^{+}mita\_mgr(\text{Hugo, EK, Emil}));$
$(sa\_\Delta^{+}mita\_mgr,\ +\Delta^{+}mita\_mgr(\text{Hugo, EK, Helga}));$
$(ip\_ic_{\Delta^{+}mita\_mgr},\ +\Delta^{+}mita\_mgr(\text{Anton, EK, Emil}));$
$(ip\_ic_{\Delta^{+}mita\_mgr},\ +\Delta^{+}mita\_mgr(\text{Anton, EK, Helga}));$
$(ip\_ic_{\Delta^{+}mita\_mgr},\ +\Delta^{+}mita\_mgr(\text{Hugo, EK, Emil}));$
$(ip\_ic_{\Delta^{+}mita\_mgr},\ +\Delta^{+}mita\_mgr(\text{Hugo, EK, Helga}));$
$(ip\_ic_{\Delta^{+}mita\_abt},\ +\Delta^{+}mita\_abt(\text{Anton, EK, Emil})) \}$

Die Ausführung der SA-Trigger-Ereignispaare $sa\_\Delta^{+}mita\_mgr$ hat einen aktualisierten Zwischenzustand der materialisierten Sicht *mita_mgr* zur Folge.

mita_mgr

Hugo	EK	Anton	Anton	EK	Emil	Anton	EK	Helga
Erna	VK	Anton	Hugo	EK	Emil	Hugo	EK	Helga
Emil	MG	Helga	Helga	MG				

Bei der Ausführung der IP-Trigger-Ereignisspaare $ip\_ic_{\Delta^+mita\_mgr}$, $ip\_ic_{\Delta^+mita\_abt}$ wird die Konsistenz der Fakten erkannt (lokal-konsistenter Zwischenzustand).

Der Aufruf der *commit_iv*-Prozedur startet den **DEFERRED-Prozeß**. Die erste Aktion ist die Löschung der 'last used'-IMMEDIATE-Fakten aus den Δ-Basisrelationen *mita_abt*, *mita_mgr*. Für die verbliebenen Δ-Fakten wird der Nettoeffekt für *mitarbeiter* und *mita_mgr* berechnet, da beide 'last used'-DEFERRED-Relationen sind ($\mathcal{LU}_D^{\Delta}$). Das Ergebnis der Nettoeffektberechnung ist die ursprüngliche Faktenmenge. Die Nettoeffektfakten werden gelöscht und wieder eingefügt, woraufhin sechs DEFERRED-Trigger-Ereignispaare gefeuert werden.

$\text{AFTER}_{mitarbeiter} :=$
$\{\ (ip\_ic_{1\Delta^+mitarbeiter},\ +mitarbeiter(\text{Anton, 10, 6000, Emil}));$
$(prop\_\Delta^+lower\_mgr_{\Delta^+mita\_mgr},\ +\Delta^+mita\_mgr(\text{Anton, EK ,Emil}));$
$(prop\_\Delta^+lower\_mgr_{\Delta^+mita\_mgr},\ +\Delta^+mita\_mgr(\text{Anton, EK, Helga}));$
$(prop\_\Delta^+lower\_mgr_{\Delta^+mita\_mgr},\ +\Delta^+mita\_mgr(\text{Hugo, EK, Emil}));$
$(prop\_\Delta^+lower\_mgr_{\Delta^+mita\_mgr},\ +\Delta^+mita\_mgr(\text{Hugo, EK, Helga}));$
$(prop\_\Delta^+lower\_mgr_{\Delta^+mitarbeiter},\ +\Delta^+mitarbeiter(\text{Anton, 10, 6000, Emil}))\ \}$

Die Ausführung des $ip\_ic_{1\Delta^+mitarbeiter}$-Trigger-Ereignispaares verursacht aufgrund der Konsistenz der Fakten kein ROLLBACK der Transaktion. Aufgrund der Ausführung der PR-Trigger-Ereignispaare werden zwei Fakten in Δ^+lower_mgr eingefügt. Die zugehörigen SA-Trigger werden gefeuert und ausgeführt. Anschließend liegt ein global-konsistenter Zustand vor.

Δ^+lower_mgr

Anton	EK	Emil	Hugo	EK	Emil

$\text{AFTER}_{mitarbeiter} :=$
$\{\ (sa\_\Delta^+lower\_mgr,\ +\Delta^+mita\_mgr(\text{Anton, EK, Emil}));$
$(sa\_\Delta^+lower\_mgr,\ +\Delta^+mita\_mgr(\text{Hugo, EK, Emil}));\ \}$

lower_mgr

Hugo	EK	Anton	Anton	EK	Emil	Hugo	EK	Emil

5.7 Vergleich mit anderen Verfahren

Für einen Vergleich mit anderen Verfahren sind zwei Schwerpunkte gewählt worden. Da die Implementierungsvorschläge SQL-basierte DB-Systeme funktional erweitern sollen, wird die Funktionalität der DB-Systeme DB2 und Oracle in Abschnitt 5.7.1 analysiert. In Abschnitt 5.7.2 werden aufgrund der Relevanz des Triggerausführungsmodells für die Entwicklung eines inkrementellen Verfahrens Lösungen basierend auf verschiedenen Trigger-Konzepten diskutiert.

5.7.1 Vergleich mit kommerziellen DB-Systemen

In Abschnitt 4.2.6 werden die Implementierungen der Integritätsbedingungen und materialisierten Sichten bei Oracle (V 8.1.x) und DB2 (V 7.x) vorgestellt.

Da **Oracle** als Hauptanwendungsgebiet für materialisierte Sichten Data Warehouses und verteilte Datenbanken sieht, können sie zu drei Zeitpunkten zum und nach dem Transaktionsende (ON COMMIT, ON DEMAND, periodisch) aktualisiert werden. Während einer Transaktion ist jedoch nur der alte Zustand der materialisierten Sicht vor Beginn der Transaktion zugreifbar. Da die Lösung für die Koordinierung der Aktualisierung zu den drei Zeitpunkten auf einem Zeitstempel mit der Transaktionsendezeit basiert, kann das hier entwickelte Konzept sehr einfach erweitert werden. Während einer Transaktion ist zur Koordinierung der IMMEDIATE- und DEFERRED-Verarbeitung in dem Attribut *pp_id* für die Δ-Fakten die PP-Nummer der zugehörigen Basisfaktenänderung gespeichert. Für die ON DEMAND und 'periodisch' zu verarbeitenden Δ-Fakten wird zum Transaktionsende die entsprechende Transaktionsendezeit gespeichert. Das Aktualisierungs- und Propagierungsverfahren basiert zwar auch auf den Grundideen der Δ-Fakten und der Spezialisierung, jedoch werden nur Sichten mit einfachen Ableitungsvorschriften inkrementell aktualisiert, und alle anderen Sichten werden rematerialisiert. Das hier entwickelte Verfahren erlaubt unabhängig von der Komplexität der Ableitungsvorschrift eine inkrementelle Aktualisierung für beliebige Sichten, was eine beachtliche Performanzsteigerung während einer Transaktion bedeuten kann.

Materialisierte Sichten mit komplexeren Ableitungsvorschriften wie rekursive Sichten, Sichten mit Mengenoperatoren oder Hierarchien von Sichten sind bei dem DB2-DB-System nicht definierbar. Da materialisierte Sichten nur auf Basisrelationen definiert werden können, ist ein Änderungspropagierungsverfahren nicht erforderlich. Anders als bei Oracle ist eine IMMEDIATE-Aktualisierung möglich. Der DEFERRED-Zeitpunkt hingegen bezeichnet einen beliebigen Zeitpunkt zu dem die REFRESH TABLE-Anweisung ausgeführt wird. Mittels dieser Anweisung kann aber eine Aktualisierung zum Transaktionsende simuliert werden. Eine Restriktion beider Implementierungen ist die Beschränkung, daß Integritätsbedingungen nur auf Basisrelationen und nicht multirelational definiert werden können. Bei Oracle wird somit die Propagierung ausschließlich für die Sichtenaktualisierung durchgeführt, die Synergieeffekte aufgrund mehrfacher Verwendung der Δ-Fakten bleiben ungenutzt.

Beide Hersteller haben für ihre Integritätsprüfungs- und Sichtenaktualisierungsverfahren die Vorteile einer in das DB-System integrierten Implementierung genutzt, so daß die Nachteile einer Präprozessor-Lösung vermieden werden konnten.

5.7.2 Vergleich mit triggerbasierten Implementierungen

In der nachfolgenden Diskussion wird die Eignung verschiedener Triggerkonzepte wie Starburst, ActLog und Chimera und IRS für die Realisierung inkrementeller Verfahren in SQL aufgezeigt, indem Lösungsskizzen für das in Abschnitt 5.5 entwickelte Verfahren entwickelt werden. Starburst ist ein viel beachtetes Trigger-Konzept, dessen Ursprünge im SQL-Kontext zu finden sind. Ceri/Widom haben zwei SQL-basierte inkrementelle Verfahren mittels Starburst-Triggern implementiert und Decker hat ein auf SQL-Triggern basieren-

des Integritätsprüfungsverfahren entworfen. ActLog hingegen ist speziell für die Aufgabe der Propagierung im Datalog-Kontext entwickelt worden und IRS für SQL-Datenbanken. Ceri/Fraternali haben die Funktionalität der Chimera-Trigger u.a. mit Oracle- und DB2-Triggern simuliert.

Starburst-Trigger:

In [CW90] haben Ceri/Widom in SQL ein auf Starburst-Triggern basierendes Integritätsprüfungsverfahren und in [CW91] ein Sichtenaktualisierungsverfahren entwickelt. Dem schließt sich eine weitere Arbeit ([CW94]) mit einem triggerbasierten Propagierungsverfahren im Datalog-Kontext an. Grundidee der Verfahren ist die Spezialisierung der Sichten/Regeln bzw. Integritätsbedingungen auf der Basis von Δ-Mengen. Statt deduktive Δ-Regeln zu generieren, werden die Δ-Fakten mittels Propagierungstriggern ermittelt bzw. mittels Integritätsprüfungstriggern geprüft. Anders als bei der Verwendung von SQL-Triggern feuern die Prozeßtrigger nicht für die Einfügungen in Δ-Relationen sondern für die Ausführung von Änderungen auf den Basisrelationen und materialisierten Sichten, was jedoch voraussetzt, daß alle Sichten materialisiert sind. Die Δ-Relationen müssen daher nicht explizit erzeugt werden. Bei der Triggerausführung liegen die geänderten Fakten in systemintern verwalteten Transitionsrelationen vor. Als Aktion der Propagierungstrigger werden keine Δ-Fakten gespeichert, sondern die induzierten Änderungen unmittelbar ausgeführt. In allen drei Verfahren wird jedoch ausschließlich zum Transaktionsende eine Verarbeitung durchgeführt.

In [CW90] gilt die Restriktion, daß Integritätsbedingungen nicht auf Sichten definiert werden können. Auf eine Propagierung von Änderungen kann somit verzichtet werden. Als Aktion im Fehlerfall ist jedoch nicht nur ein ROLLBACK vorgesehen. Während des Generierungsprozesses für die IP-Trigger hat der Anwender die Gelegenheit, im Aktionsteil fehlerkorrigierende Aktionen zu programmieren ('constraint maintenance, integrity repair'). Ihre Erkenntnisse zur Spezialisierung von Bedingungen mit SQL-Aggregatfunktionen und weiteren SQL-Operatoren wie OR, IN, INTERSECT, MINUS könnten unmittelbar auf ein entsprechend erweitertes NSQL übertragen werden.

Für ein Propagierungsverfahren zur Sichtenaktualisierung diskutieren Ceri/Widom in [CW91] die Syntaxanalyse der SQL-Sichten sehr ausführlich. Sie verwenden dabei anders als in dieser Arbeit nicht das aus dem Datalog-Kontext bekannte Vokabular, wodurch die Analogien zwischen den Relevanzanalysen in SQL und Datalog nur schwer erkennbar werden. Ein wichtiges Kriterium für die Spezialisierung von Sichten ist das Vorhandensein von Schlüsselattributen. Im wesentlichen werden nur Änderungen von Relationen inkrementell propagiert, deren Schlüsselattribute auch in der Attributliste der Sicht auftreten, und deren Werte unverändert sind. Aufgrund dieser Restriktion kann der Effektivitätstest auf die Duplikatkontrolle der während der Transaktion geänderten Fakten beschränkt werden. Für Änderungen, die nicht inkrementell verarbeitet werden können, werden Rematerialisierungstrigger generiert. Da keine rekursiven Sichten zugelassen sind, gewährleistet diese eingeschränkte Anwendung der Effektivitätstests, daß die spezialisierten Δ-Ausdrücke stratifizierbar sind

Zur Lösung des Stratifikationsproblems aufgrund von Effektivitätstests bei Δ-Regeln für induzierte Löschungen leiten Ceri/Widom in Datalog ([CW94]) nur ableitbare Löschun-

gen ab und führen sie aus. In einem zweiten Schritt werden die gelöschten Fakten, für die alternative Ableitungswege bestehen, wieder eingefügt ('re-insert'-Trigger). Bei den hier entwickelten Verfahren wird dieses Stratifikationsproblem wie bei [Grie97] durch die Verwendung direkter und indirekter Transitionssichten gelöst (Abschnitt 4.2.4).

Die Ausführung eines **DEFERRED-Prozesses** wird sehr gut durch das Ausführungsmodell der Starburst-Trigger unterstützt. Sie feuern zum Transaktionsende für den Nettoeffekt der Basisfaktenänderungen und werden mengenorientiert für eine Änderungsanweisung ausgeführt. Der reihenfolgeunabhängige Nettoeffekt wird von der Starburst-Triggerkomponente berechnet, während der Nettoeffekt in SQL explizit ermittelt werden muß. Zum Zeitpunkt der Triggerausführung liegen die Relationen im neuen Zustand vor, im Aktions- und Bedingungsteil der Trigger kann auf systemverwaltete Transitionsrelationen zugegriffen werden, die die Mengen der Nettoeffektänderungen enthalten. Der alte Zustand der Relationen wird mittels Transitionssichten simuliert. Die Priorität eines Triggers wird in der PRECEDES-Klausel mittels einer partiellen Ordnung spezifiziert. Die stratumbezogene Ausführungsreihenfolge für die Prozeßtrigger kann ebenso einfach definiert werden wie die Ausführung der Trigger der verschiedenen Klassen während der Bearbeitung der Sichten eines Stratums. Wie intuitiv der Start einer DEFERRED-Verarbeitung erfolgen kann, wenn Trigger zum Transaktionsende ausgeführt werden, wird im Vergleich zu den SQL-triggerbasierten Verfahren sehr deutlich.

Eine **IMMEDIATE ausgeführte Propagierung** und Verarbeitung wird von Ceri/ Widom nicht beschrieben und läßt sich, da Starburst-Trigger nicht zum entsprechenden Zeitpunkt ausgeführt werden, auch nicht unmittelbar realisieren. In [CW91] wird eine Funktionserweiterung um 'rule assertion points' in Aussicht gestellt. Anwender können 'rule assertion points' zu jedem beliebigen Zeitpunkt erzeugen. Analog zum Verhalten am Transaktionsende wird zu einem 'rule assertion point' für alle Änderungen seit dem letzten Punkt der Nettoeffekt berechnet und die Trigger werden ausgeführt. Zu einem 'rule assertion point' ist eine Differenzierung zwischen IMMEDIATE und DEFERRED auszuführenden Triggern nicht möglich. Mittels eines derart erweiterten Starburst-Ausführungsmodells ist entweder eine Verarbeitung zum IMMEDIATE- oder zum DEFERRED-Zeitpunkt möglich. Ein Verfahren zur IMMEDIATE- und DEFERRED-Verarbeitung wäre ähnlich aufwendig zu simulieren wie mit SQL-Triggern.

Integritätsprüfung von Decker mittels Triggern:

Decker stellt in [Dec01] eine SQL-triggerbasierte Simulation der Integritätsprüfung ('soundcheck') vor. Zugelassen sind Formeln mit der Ausdruckskraft von SQL3-CHECK-Bedingungen, die auf Basisrelationen definiert sind, so daß eine Änderungspropagierung nicht erforderlich ist. Als Änderungsanweisungen werden Einfügungen und Löschungen betrachtet. Die Relevanzanalyse und Spezialisierung wird für Ausdrücke der Prädikatenlogik 1. Ordnung diskutiert. Anschließend werden die spezialisierten Formeln ('update constraint') in bereichskalkülbasierte 'range form'-Formeln transformiert (*rf*-Funktion), die einen Zwischenschritt zur Generierung einer SQL-Formel (*sql*-Funktion) und eines Integritätsprüfungstriggers darstellen. Die Integritätsprüfungstrigger feuern für die Ausführung von Einfügungen bzw. Löschungen in Basisrelationen. Es werden keine Δ-Relationen definiert, sondern auf die systeminternen Transitionsrelationen der Triggerkomponente zugegriffen. Im Aktionsteil der Integritätsprüfungstrigger wird die spezialisierte Inte-

gritätsbedingung ausgewertet und das Ergebnis interpretiert. Das lokale Zurückrollen bei einer Konsistenzverletzung wird durch eine ROLLBACK-Anweisung ausgelöst.

Das Triggerausführungsmodell diskutiert Decker nicht ausführlich und bezeichnet die verwendeten Trigger als einen 'gemeinsamen Nenner'[10] der in kommerziellen DB-Systemen zur Verfügung stehenden Trigger. Den Beispielen kann entnommen werden, daß die Trigger nach der Ausführung einer Änderungsanweisung mengenorientiert feuern, so daß zum Zeitpunkt der Prüfung bereits der neue Zustand vorliegt und damit Transitionssichten nicht erforderlich sind. Mittels der Trigger wird eine IMMEDIATE-Prüfung realisiert. Die Anforderungen einer DEFERRED-Prüfung bzw. von Prüfungen zu beiden Zeitpunkten werden nicht erörtert. Für die nur für Basisrelationen formulierbaren Integritätsbedingungen ist eine Änderungspropagierung nicht erforderlich, so daß es auch nicht notwendig ist, daß Triggerausführungsmodell detaillierter zu erörtern. Aufgrund der fehlenden Detail-Informationen kann jedoch eine Lösungsskizze für das hier entwickelte Verfahren nicht entworfen werden.

Integritätsprüfung von Reinert mittels IRS-Triggern:

Reinert stellt in seiner Dissertation ([Rei96]) ein aktives Integritätsregelsystem (IRS) zur Simulation des SQL-Integritätsprüfungskonzepts mit ASSERTIONs und FOREIGN KEY-Constraints vor. Seine Lösung basiert nicht auf inkrementellen Ansätzen sondern konzentriert sich auf die Anforderungen, die sich für ein Triggerausführungsmodell ergeben. IRS-Regeln sind ECA-Regeln, die für die Ausführung einer Änderungsanweisung auf einer Relation (Basisrelation oder Sicht) oder auch bei Eintritt eines Transaktionsereignisses (Start, Ende und Abbruch einer Transaktion), aktiviert werden. Vor der Ausführung der Änderungsanweisungen des Aktionsteils kann eine Bedingung geprüft werden. Eine weitere Bedingung kann zum Ende der Triggerausführung oder einem späteren Zeitpunkt evaluiert werden, z.B. zur Kontrolle der Konsistenz nach Ausführung fehlerkorrigierender Änderungen im Aktionsteil. Während der Triggerausführungen kann auf die Fakten in Transitionsrelationen und -variablen zugegriffen werden.

Zu einem Ereignis wird die Menge der aktivierten Trigger 'logisch gleichzeitig' ausgeführt, im nächsten Schritt werden die durch die Triggerausführungen in der vorangegangenen Phase aktivierten Trigger ausgeführt. Da keine Prioritäten für IRS-Trigger definierbar sind, müßte z.B. im Rahmen eines 'bottom up' ausgeführten Propagierungsverfahrens für die Steuerung der Ausführungsreihenfolge von PR-, IP- und SA-Triggern während eines Stratums alternative Lösungen entwickelt werden. Ebenfalls müßte explizit die Ausführung eines zum Stratum i aktivierten Prozeßtriggers zu einem Stratum $i + x$ $(x > 0)$ gesteuert werden.

IRS-Regeln vermeiden jedoch auch einige der Nachteile der SQL3-Trigger. So können sowohl die Bedingungen (C) als auch die Aktionen (A) nicht nur IMMEDIATE sondern auch DEFERRED angewendet werden, wodurch eine aufwendige Simulation der DEFERRED-Ausführung wie bei SQL3-Trigger nicht erforderlich ist. Dadurch, daß IRS-Trigger auch für Sichten definiert werden können, kann bei einer IRS-basierten Lösung die Optimierungsidee berücksichtigt werden, für jede Δ-Relation explizit zu entscheiden, ob sie als

[10] [Dec01] 5: 'In general, we assume that SQL triggers are of the following form (which essentially is a common denominator of the usual appearance of triggers in commercial SQL database systems.')

Basisrelation oder Sicht erzeugt werden soll.

IRS ist sehr aufgabenspezifisch entwickelt worden und ist insbesondere unter Beachtung des SQL-Transaktions- und Integritätskonzepts sehr gut zur Simulation der Integritätsprüfung in SQL geeignet. Als alternative Anwendungsmöglichkeit diskutiert Reinert kurz, die 'view update'-Problematik, nicht jedoch die Änderungspropagierung und Aktualisierung materialisierter Sichten. Da IRS nicht als Triggerkomponente in kommerziellen DB-Systemen zur Verfügung steht, kann eine IRS-basierte Implementierung auch keine funktionale Erweiterung bestehender SQL-basierter DB-Systeme darstellen.

ActLog-Trigger:

Griefahn entwickelt in [Grie97] die aktive DB-Sprache ActLog (Abschnitt 2.1.4), deren Leistungsfähigkeit anhand eines Änderungspropagierungsverfahrens als Anwendungsbeispiel aufgezeigt wird. Die abgeleiteten Δ-Fakten können zur inkrementellen Integritätsprüfung und Sichtenaktualisierung verwendet werden. Im Rahmen der Spezialisierung von Datalog-Regeln werden zunächst Δ-Regeln definiert. Anschließend werden Transformationsvorschriften zur Abbildung der Datalog-Regeln auf ActLog-Trigger formuliert.

Neben einem grundlegenden Lösungsansatz beschäftigt sich Griefahn mit einer effizienten Auswertung von Effektivitätstests auf der Basis von 'Magic Sets'-Transformationen der Δ-Sichten. Für die Auswertung von Regelmengen, die aufgrund der 'Magic Sets'-Transformation unstratifizierbar geworden sind, definiert Griefahn einen alternierenden Fixpunkt. Für die Stratifikationsprobleme, die aufgrund von Effektivitätstests und rekursiven Regeln auftreten können, wird eine Lösung basierend auf inkrementellen Transitionsregeln vorgeschlagen (Abschnitt 4.2.4).

Anders als bei den PR- und SA-Triggern wird statt zwei Triggertypen nur ein Propagierungs- und Aktualisierungstrigger ('active materialization rule') verwendet. Die abgeleiteten induzierten Änderungen werden unmittelbar auf den Sichten ausgeführt. Dies setzt voraus, daß alle Sichten materialisiert vorliegen. Δ-Basisrelationen müssen nicht definiert werden. Da ActLog-Trigger wie die SQL-ROW-Trigger tupelorientiert ausgeführt werden, und somit Variablenbindungen für die zu propagierende Änderung aufgrund des Ereignisses gegeben sind, wird die Formel im Regelrumpf analog zu der SQL-Ableitungsvorschrift spezialisiert.

Das Ausführungsmodell der ActLog-Trigger unterstützt ebenfalls einen **DEFERRED-Prozeß** sehr intuitiv und sehr weitreichend. Zum Transaktionsende werden für alle Ereignisse der Transaktion die Trigger gefeuert, indem Trigger-Ereignispaare gebildet werden (tupelorientiert). Die Trigger-Ereignispaare werden gemäß ihrer Prioritäten geordnet und die Menge der Trigger-Ereignispaare mit der höchsten Priorität werden ausgeführt. Die aufgrund dieser Triggerausführungen neu gebildeten Trigger-Ereignispaare werden entsprechend ihrer Prioritäten in die Menge aller Trigger-Ereignispaare der Transaktion eingeordnet, so daß analog zu der Abarbeitung der AFTER-Liste in SQL die Prozeßtrigger stratumbezogenen ausgeführt werden können. Da Trigger nur für 'safe' Ereignisse ausgeführt werden, und die Sicherheit vor der Ausführung jeweils für die Menge der Trigger-Ereignispaare mit der höchsten Priorität kontrolliert wird, ist gewährleistet, daß nur Nettoeffektfakten propagiert werden.

Die Ausführungszeitpunkte für Trigger können zwar grundsätzlich zu einem beliebigen Zeitpunkt während der Transaktion gesetzt werden, zu jedem dieser Zeitpunkte würde das Verfahren jedoch wie zum Transaktionsende ausgeführt. Eine Differenzierung in verschiedene Integritätsbedingungen und Sichten, die zu verschiedenen Zeitpunkten bearbeitet werden, ist ebensowenig möglich wie eine IMMEDIATE- und DEFERRED-Änderungspropagierung. Beim Versuch, auch zum **IMMEDIATE-Zeitpunkt** einen Prozeß durchzuführen, treten die gleichen Probleme auf wie bei Starburst, so daß für die Entwicklung eines entsprechenden Verfahrens auch auf die gleichen Konzepte und Techniken zurückgegriffen werden müßte wie bei Starburst.

Die beiden Lösungsskizzen für Verfahren, die den konzeptionellen Anforderungen eines SQL-basierten DB-Systems genügen, zeigen, daß ActLog- und Starburst-Trigger ebenso wie SQL-Trigger ein Verfahren mit zwei Prozeßzeitpunkten nur beschränkt unterstützen. Während bei SQL-Triggern im wesentlichen der Start des DEFERRED-Prozesses aufwendig zu realisieren ist, ist es bei ActLog und Starburst die Realisierung des IMMEDIATE-Prozesses. Für ActLog ist dieses Ergebnis naheliegend, da ActLog speziell für die Realisierung von DEFERRED-Prozessen im Datalog-Kontext entwickelt wurde. Starburst- und SQL-Trigger hingegen sind anwendungsunabhängig als funktionale Erweiterung relationaler SQL-basierter DB-Systeme entwickelt worden. Ein wesentlicher Vorteil der ActLog- und Starburst-Ausführungsmodelle ist die implizite und an das zugrunde liegende Transaktionsmodell angepaßte Berechnung des Nettoeffekts der Änderungen zum Transaktionsende. Bei Verfahren im SQL-Kontext muß diese aufwendige Berechnung explizit durchgeführt werden.

Implementierung von Chimera in DB2- und Oracle-DB-Systemen:

Zur Unterstützung des Entwurfs anspruchsvoller DB-Applikationen auf der Basis von Objekten, deduktiven Regeln und Triggern präsentieren Ceri/Fraternali in [CF97] eine objekt-orientierte Entwurfsmethode: die IDEA-Methode [11]. Als objekt-orientierte, deduktive und aktive DB-Sprache wird Chimera[12] verwendet. Die Besonderheiten, die sich im weiteren aus dem zugrunde liegenden objekt-orientierten Modell ergeben, werden nicht erörtert, da dies über das Thema dieser Arbeit hinausgehen würde. Gegenstand der nachfolgenden Vergleiche sind die beiden Punkte der Propagierungs-, Sichtenaktualisierungs- und Integritätsprüfungsverfahren in Chimera und die Implementierungen dieser Verfahren in DB-Systemen wie u.a. Oracle und DB2.

In [GrMa94] und [CF97] Kap. 12 wird die Strategie der inkrementellen Änderungspropagierung in Chimera erörtert. Das Propagierungsverfahren ist die zentrale Komponente zur Bestimmung der induzierten Änderungen sowohl für die Integritätsprüfung wie für die Sichtenaktualisierung und wird mittels Chimera-Triggern implementiert. Für die Änderungspropagierung wird in [GrMa94] die Chimera-Syntax auf deduktive Re-

[11] Das IDEA-Projekt ist ein Teilprojekt des von der Europäischen Union von 1992 bis 1996 geförderten Esprit-Projekts, welches mit industriellen Partnern und verschiedenen europäischen Forschungsinstituten wie dem ECRC in München, dem INRIA in Paris und den Universitäten in Mailand und Bonn durchgeführt wurde.

[12] Die aktive DOOD Chimera wurde 1992 von Elisa Bertino, Stefano Ceri und Rainer Manthey entworfen ([GrMa94], [CF97]).

geln beschränkt, die mit stratifizierbaren Datalog-Regeln mit Negation und ohne Duplikaten vergleichbar sind. Deduktive Chimera-Regeln werden auch zur Formulierung von Integritätsbedingungen verwendet. Propagierte Änderungsarten sind Einfügungen, Löschungen und Modifikationen. Da im objekt-orientierten Modell eine eindeutige und unveränderliche Objekt-ID (OID) zur Verfügung steht, stellt die Verwendung des Ansatzes von Olivé et.al. ([UO92])zur Propagierung von Modifikationen auf der Basis von Schlüsselattributen anders als im relationalen Modell keine Einschränkung dar.

Das Triggerausführungsmodell wird hier nur gemäß den Beschreibungen in [GrMa94] skizziert. Chimera-Trigger sind ECA-Trigger, die mengenorientiert für eine Änderungsanweisung oder Anfrage bzgl. einer Klasse (E) gefeuert werden. Gefeuerte Trigger werden entweder IMMEDIATE nach einer feuernden Aktion oder DEFERRED zum Transaktionsende ausgeführt. Aus der Menge der aktivierten Trigger des aktuellen Ereignisses bzw. des Transaktionsendes wird ein Trigger mit der höchsten Priorität ausgewählt und ausgeführt. Als erster Schritt der Triggerausführung wird die Bedingung (C) überprüft, und für jedes Objekt, das diese Bedingung erfüllt, wird der Triggeraktionsteil (A) ausgeführt. Durch den Triggeraktionsteil können weitere Trigger gefeuert werden, die entsprechend ihrer Priorität in die Liste der aktivierten Trigger eingeordnet werden.

Das **Änderungspropagierungsverfahren** basiert wie bei den SQL-basierten inkrementellen Verfahren auf der Definition von '*delta classes*', welche in ihrer Funktion den Δ-Basisrelationen entsprechen. Für jede Klasse und jedes Attribut wird eine '*delta class*' erzeugt. Analog zu den PL-Triggern werden für die Protokollierung der Änderungen in den '*delta classes*' '*initial update propagation*'-Trigger erzeugt. Die Spezialisierung der Δ-Regeln im objekt-orientierten Kontext orientiert sich an den Spezialisierungsstrategien, wie sie für relationale DB-Systeme entwickelt wurden. Die Lösung des Stratifikationsproblems bei der Propagierung effektiver Änderungen basiert auf der Idee des zur Laufzeit sehr aufwendigen und zeitintensiven 'Re-Insert'-Algorithmus von Ceri/Widom in [CW94]. Bei Chimera ist wie bei SQL eine Transaktion eine Sequenz von Änderungen. Für die Berechnung des Nettoeffekts von Änderungen kann in der Formel der Triggerbedingung (C) eine spezielle von Chimera bereitgestellte Funktion aufgerufen werden. Auf eine Konsistenzverletzung bei einer IMMEDIATE-Prüfung wird jedoch statt mit einem lokalen ROLLBACK mit dem Zurückrollen der gesamten Transaktion reagiert[13].

Die für Chimera und SQL entwickelten inkrementellen Verfahren basieren auf sehr ähnlichen Grundideen, wie z.B. die Trigger zur Protokollierung der Δ-Fakten von Basisfaktenänderungen. Doch da Chimera eine objekt-orientierte Sprache ist, müssen die Verfahren vielen Anforderungen genügen, die von denen im relationalen Kontext sehr verschieden sind. Auch die Unterschiede im Triggerausführungsmodell erlauben keine unmittelbare Anwendung der Chimera-Lösung in SQL. Die Möglichkeit, Trigger zum Transaktionsende für den Nettoeffekt der Änderungen der Objekte einer Klasse ausführen zu lassen, vereinfacht die Ausführungssteuerung des DEFERRED-Prozesses in Chimera wesentlich. Aufgrund der beiden Triggerausführungszeitpunkte ist zudem die Kontrolle, welche Änderungen zu welchem Prozeßzeitpunkt zu propagieren sind, vollständig der Chimera-Triggerkomponente überlassen.

[13] [CF97] 2.1.3, S. 21

Grundidee der Implementierungen von Chimera in DB-Systemen kommerzieller Hersteller ist, daß die Funktionalität soweit wie möglich durch Chimera-DB-Objekte bereit gestellt wird, und dann diese Chimera-Objekte auf die Objekte des DB-Systems abgebildet werden. Somit werden keine triggerbasierten Verfahren speziell für Oracle- und DB2-Trigger entwickelt, sondern es werden prinzipielle Abbildungsvorschriften für Chimera-Trigger, -Regeln, -DB-Prozeduren und alle weiteren Chimera-DB-Objekte formuliert. Für den Vorteil einer allgemeinen Abbildungsvorschrift wird ggf. auf die Ausnutzung aufgabenspezifischer Optimierungsmöglichkeiten verzichtet.

Die **Chimera-Integritätsbedingungen** werden soweit wie möglich auf die bei der DB2 und Oracle gegebenen Basisrelationen-Constraints abgebildet('built-in constraints': NOT NULL, FOREIGN KEY, PRIMARY KEY, UNIQUE KEY). Allgemeine Integritätsbedingungen ('generic integrity constraints') werden auf spezialisierte Integritätsprüfungstrigger abgebildet. Die Spezialisierung erfolgt nach den im Datalog- und RA-Kontext bekannten Strategien. Als Δ-Relationen werden die für jede Klasse und jede Änderungsoperation generierten '*delta classes*' verwendet. Neben statischen Integritätsbedingungen können auch transitionale Integritätsbedingungen formuliert werden. Wie bereits erörtert, können transitionale Bedingungen aufgrund der Verfügbarkeit von OLD- und NEW-Transitionsvariablen ohne Einschränkungen auf Integritätsprüfungstrigger abgebildet werden. Integritätsbedingungen sind auch für abgeleitete Klassen definierbar, nicht jedoch für Sichten.

Zentrale Schwierigkeit bei der Abbildung der Chimera-Trigger auf Oracle-Trigger ist die Simulation der auf Prioritäten und zeitlichen Verzögerungen basierende Ausführungssteuerung der Chimera-Trigger ('breadth first'), denn Oracle-BEFORE- und -AFTER-Trigger werden beide wie SQL-BEFORE-Trigger unmittelbar und ohne jede zeitliche Verzögerung ausgeführt ('depth first'). Für die Simulation werden für die Bedingungs- (C) und Aktionsteile (A) der Chimera-Trigger Oracle-DB-Prozeduren erzeugt (CA-Prozedur). Damit diese CA-Prozedur zu den erforderlichen Zeitpunkten ausgeführt wird, wird für jede CA-Prozedur ein Trigger generiert, der für ein 'künstliches' Ereignis feuert. 'Künstlich' meint, daß der Trigger nicht für ein der Aufgabenstellung angemessenes Ereignis feuert (Propagierung: Einfügung in Δ-Relationen) sondern für das Setzen einer entsprechenden Ausführungskennung in einer Trigger-Steuerungsrelation, die anzeigt, welcher Trigger mit welcher ID als nächstes auszuführen ist. Die Triggerausführungssteuerung wird durch eine prozedurale Steuerung überlagert ('meta triggering').

Diese Technik wird auch für die Ausführung von Oracle- und DB2-Triggern zum DEFERRED-Zeitpunkt angewendet. Anders als beim SQL-basierten Verfahren, in welchem durch eine Art 'Wiederholung' des feuernden Ereignisses die DEFERRED-Trigger zum Transaktionsende aktiviert werden, wird ihre Ausführung bei den Chimera-Implementierungen prozedural gesteuert. Einzige Aufgabe der DEFERRED auszuführenden Trigger ist die Ausführung der Prozedur, die im Aktionsteil programmiert ist. Somit können die Vorteile, die die triggerbasierte Implementierung des SQL-basierten Verfahrens motiviert haben, nicht genutzt werden. Die Kontrolle des Prozeßendes muß von der Prozedur geleistet werden, ebenso wie die Kontrolle, welche Trigger relevant sind für eine Änderung, und auch die Kontrolle, ob bei rekursiven Regeln weiterhin neue Fakten abgeleitet werden. Da die Ausführungssteuerung der IMMEDIATE ausgeführten Chimera- und DB2-Trigger übereinstimmt, können die Chimera-Trigger direkt auf DB2-Trigger abgebildet werden.

Als Fazit kann festgehalten werden, daß aufgrund der Objekt-Orientierung von Chimera das inkrementelle Propagierungsverfahren mit Integritätsprüfung und Sichtenaktualisierung nicht nach SQL übertragen werden kann. Da aber das Propagierungsverfahren für Chimera entwickelt wurde, und dann nur die einzelnen Chimera-DB-Objekte auf Oracle- oder DB2-DB-Objekte abgebildet werden, ergeben sich trotz vieler analoger Konzepte insbesondere in der Ausführungssteuerung und der Modellierung der Prozeßtrigger Unterschiede gegenüber dem SQL-basierten Verfahren, das unter Ausnutzung der Besonderheiten der von SQL zu Verfügung gestellten Funktionalität entwickelt worden ist.

Kapitel 6

Zusammenfassung und Perspektiven

Für die Anwendung im SQL-Kontext ist ein Verfahren zur inkrementellen Änderungspropagierung entwickelt und in einzelnen Schritten funktional um Ansätze zur effizienten Sichtenaktualisierung und Integritätsprüfung erweitert worden. Die auf SQL-Triggern basierenden Implementierungsansätze gewährleisten zur Laufzeit die Ausführung des Prozesses als 'bottom up'-Fixpunktprozeß. Die für die Verfahrensausführung erforderlichen DB-Objekte werden von einem Präprozessor generiert.

Zusammenfassung

In Abschnitt 2.1 wird die Regelsprache Datalog um eine Änderungssprache erweitert. Die Datalog-DML und das zugehörige Transaktionskonzept entsprechen in ihren zentralen Eigenschaften den Konzepten, die den in Kapitel 4 vorgestellten inkrementellen Verfahren zugrunde liegen. SQL3 wird in Abschnitt 2.2 mit einer für das Thema der Arbeit relevanten Funktionalität definiert. Rekursive Sichten sind formulierbar, jedoch keine Aggregatfunktionen. Für beide Sprachen werden aktive Regelkonzepte erörtert: ActLog-Trigger für Datalog und SQL3- und Starburst-Trigger für SQL.

Die in Abschnitt 3.1 normalisierte DB-Sprache NSQL ist gegenüber dem im vorangegangenen Abschnitt definierten SQL3 semantisch nicht weiter eingeschränkt. Die ausgewählten SQL3-Anweisungen und -Operatoren verdeutlichen die Analogien zwischen SQL3 und Datalog. Mit den in Abschnitt 3.2 formulierten Funktionen können Datalog-Regeln, -Anfragen und -Integritätsbedingungen direkt auf NSQL-Ausdrücke abgebildet werden und umgekehrt. Bislang sind nur Vorschriften zur Transformation von SQL-Ausdrücken in Ausdrücke Relationaler Algebra definiert worden und für Datalog Vorschriften zur Transformation ins Bereichskalkül.

Das in Abschnitt 4.1 im Datalog-Kontext definierte generische Verfahren gibt einen Einblick in die Gesamtzusammenhänge eines inkrementellen Verfahrens zur Änderungspropagierung mit Integritätsprüfung und Sichtenaktualisierung. Die Ableitung induzierter Änderungen erfolgt mittels passiver Datalog-Regeln. Die Ausführungssteuerung der Propagierung als 'bottom up'-Fixpunktprozeß, die Aktualisierung der Sichten und die Interpretation der Ergebnisse der Integritätsprüfungen sind Aufgaben einer Steuerungsprozedur. Die Spezialisierung der Ableitungsvorschriften und Integritätsbedingungen basiert auf der Methode von Griefahn ([Grie97]). Dabei werden wie auch bei dem nachfolgend entwickelten SQL-basierten Verfahren zur Lösung des Stratifikationsproblems bei der Pro-

pagierung effektiver Änderungen die 'incremental transition rules' von Griefahn verwendet. Gegenüber den 'DRed'- und 'Re-Insert'-Algorithmen von Gupta et.al. ([GMS93]) und Ceri/Widom ([CW94]) wird somit vermieden, zuerst eine Obermenge effektiv zu löschender Fakten zu entfernen und anschließend die gelöschten Fakten mit alternativen Ableitungswegen wieder einzufügen.

Ausgehend von dem generischen Verfahren werden in Abschnitt 4.2 für die grundlegenden Eigenschaften eines inkrementellen Verfahrens (wie die Granularität induzierter Änderungen, die Auswertungsstrategie, die Implementierung der Δ-Mengen, die Transitionsregeln und die Relevanzanalyse) alternative Lösungen mit ihren Vor- und Nachteilen erörtert. Die Diskussion der Implementierungen in DB-Systemen wie DB2 und O8i von IBM und Oracle zeigt, daß dort zwar die gleichen grundlegenden Konzepte wie z.B. die Spezialisierung der Sichtdefinitionen angewendet werden, sie zeigt aber auch, welchen starken Einschränkungen sowohl die Konzepte materialisierter Sichten als auch die Integritätskonzepte unterliegen.

Die Integration der in Abschnitt 4.3 erörterten Lösungen für die Behandlung von Aggregatfunktionen, Duplikaten, Modifikationen und transitionalen Integritätsbedingungen in das generische Verfahren würde eine echte funktionale Erweiterung darstellen, ebenso wie die Strategien zur Integritätserhaltung, die beiden Aktualisierungszeitpunkte 'periodisch' und ON DEMAND und weitere orthogonale Optimierungsideen. Die dort definierten Lösungen können ohne wesentliche Einschränkungen und ohne viel Aufwand in das generische Verfahren und bei Anpassung an die SQL-Syntax auch in das SQL-basierte Verfahren integriert werden. Da das von Olivé, Urpí und Teniente et.al. ([UO92], [UO94], [TO95], [MT00]) entwickelte Verfahren zur Propagierung von Modifikationen auf dem Spezialfall des Vorhandenseins von Schlüsselattributen basiert, wird für den Datalog-Kontext ein erweitertes Konzept zur Ableitung induzierter Modifikationen formuliert, welches dann auch in den NSQL-Kontext übertragen wird. Die grundlegenden Vorschriften für NSQL zeigen die Bedeutung des Auftretens der Attribute in den verschiedenen SELECT-, FROM-, WHERE-Klauseln und den Unteranfragen eines Anfrageausdrucks. Für eine Anwendung in SQL3 müssen diese grundlegenden Erkenntnisse nur noch auf den Fall beliebiger Schachtelungstiefen von Unteranfragen übertragen werden.

Nach ausführlichen Diskussionen verschiedener Alternativen in Abschnitt 5.1 werden in Kapitel 5 Präprozessor-Lösungen für triggerbasierte Implementierungsansätze inkrementeller Verfahren entwickelt. Für die Bewertung der Eignung der verschiedenen SQL-Triggertypen werden jeweils Lösungen skizziert, wobei deutlich wird, daß die AFTER-Trigger die Ausführung iterativer Fixpunktprozesse sehr weitgehend unterstützen. Als problematisch haben sich jedoch zwei Punkte erwiesen. Da zum Transaktionsende keine Trigger feuern bzw. ausgeführt werden, ist der Start der DEFERRED-Verarbeitung (Aufruf der *commit_iv*-Prozedur) im Gegensatz zur IMMEDIATE-Verarbeitung nicht transparent für den Benutzer. Die Unvereinbarkeit der Integritätsprüfungszeitpunkte triggerausgeführter Änderungsanweisungen mit dem Ausführungszeitpunkt von AFTER-Triggern hat sich ebenfalls als Problem erwiesen. Zudem können einige Optimierungsstrategien aus Kapitel 4 wie die relationsspezifische Entscheidung für virtuelle oder materialisierte Δ-Sichten aufgrund der triggerbasierten Implementierung nicht angewendet werden.

Für ein Propagierungsverfahren werden in Abschnitt 5.2 die SQL-spezifischen Anforderungen und Optimierungsmöglichkeiten erörtert, die im wesentlichen aus dem SQL-Transaktions- und Integritätskonzept resultieren. Das dabei entwickelte Propagierungskonzept basiert auf den beiden Propagierungszeitpunkten IMMEDIATE und DEFERRED. Für die DEFERRED-Verarbeitung steht die Erkenntnis im Vordergrund, daß die reihenfolgeabhängigen Nettoeffekte der SQL-Transaktionen explizit berechnet werden müssen. Für dieses SQL-Propagierungskonzept wird ein triggerbasierter Implementierungsansatz aufgezeigt und ein Präprozessor definiert.

Dieses Propagierungsverfahren wird für die Aufgabenstellungen der Integritätsprüfung, der Sichtenaktualisierung und für beide Aufgaben zusammen erweitert wie auch die Aufgaben des Präprozessors (Abschnitte 5.3, 5.4, 5.5). Es wird das NSQL-Integritätskonzept, das sich auf SQL3-ASSERTIONs beschränkt, zugrunde gelegt. Da Vorgaben von Seiten des SQL-Standards fehlen, wird für materialisierte Sichten in SQL ein eigenes Konzept entwickelt. Wesentliche Eigenschaft der materialisierten Sichten sind die beiden Aktualisierungszeitpunkte IMMEDIATE und DEFERRED. Die Abläufe beim Schemaentwurf und bei der Ausführung einer Transaktion werden anhand eines Beispiels in Abschnitt 5.6 veranschaulicht. Abschließend wird die Leistungsfähigkeit der hier definierten Integritäts- und Sichtkonzepte mit der von DB-Systemen kommerzieller Hersteller verglichen. Ebenso wird der SQL-triggerbasierte Implementierungsansatz mit anderen triggerbasierten Lösungen von Ceri/Widom ([CW90], [CW91]), Griefahn ([Grie97]), Decker ([Dec01]), Reinert ([Rei96]) und Ceri/Fraternali ([CF97]) verglichen, wobei sich aufgrund unterschiedlicher Triggerausführungsmodelle wesentliche Unterschiede ergeben.

Ergebnisse

Aufgrund der Vielschichtigkeit der Aufgabenstellung lassen sich mehrere wesentliche Ergebnisse zusammenfassen.

- Als Ergebnis der Sprachanalysen kann festgestellt werden, daß aufgrund der Zulässigkeit von Rekursion für SQL3 ein normalisiertes NSQL identifiziert werden kann, welches eine zu Datalog vergleichbare Semantik aufweist. Bei entsprechender Auswahl der in NSQL zulässigen Befehle und Operatoren sind die Analogien zwischen NSQL- und bereichsbeschränkten Datalog-Ausdrücken gut erkennbar und die erforderlichen Transformationsschritte intuitiv verständlich.

- Da insbesondere die beiden Prozeßzeitpunkte IMMEDIATE und DEFERRED in SQL zu beachten sind, erfordert die Anwendung der bereits in Datalog und Relationaler Algebra entwickelten inkrementellen Verfahren eine Reihe von Anpassungen und Erweiterungen. Für das SQL-Transaktions- und Integritätskonzept ist ein Basisverfahren entwickelt worden, das grundlegende, inkrementelle Strategien zur effizienten Integritätsprüfung und Sichtenaktualisierung auf der Basis der Methode von Griefahn ([Grie97]) umsetzt. Für das Problem der Propagierung induzierter Modifikationen in SQL sind umfassende Propagierungsvorschriften formuliert worden.

- Das dem SQL-basierten inkrementellen Verfahren zugrunde gelegte Konzept materialisierter Sichten stellt eine konzeptionelle Erweiterung des SQL3-Standards dar. Aufgrund der beiden Aktualisierungszeitpunkte IMMEDIATE und DEFERRED werden insbesondere zwei verschiedene Anwendungsfälle sehr gut unterstützt. Dies ist

zum einen die Zugriffsoptimierung von Anfragen auf aktualisierte Sichten während der laufenden Transaktion (IMMEDIATE) und zum anderen die Verwendung in Data Warehouses und in verteilten Datenbanken (DEFERRED).

- Die Implementierung der SQL-basierten inkrementellen Verfahren in den DB-Systemen DB2 und Oracle würde die Integritätskomponenten dieser Systeme um multirelationale und sichtenbasierte Integritätsbedingungen erweitern, so daß sie den Anforderungen des SQL-Standards entsprechen würden. Die in den DB-Systemen realisierten Konzepte materialisierter Sichten würden u.a. um die Möglichkeiten zur Definition sowohl rekursiver Sichten als auch von Hierarchien von Sichten ergänzt. Kommerzielle DB-Systeme würden zudem um eine Änderungspropagierungskomponente erweitert, deren Ergebnisse auch für weitere funktionale Systemerweiterungen verwendet werden können. Die Ableitung induzierter Änderungen könnte dann als feuerndes Ereignis für sichtenbasierte Trigger definiert werden. Die Information über alle induzierten Änderungen könnte zudem für Anwender als Entscheidungsgrundlage darüber dienen, ob eine Basisfaktenänderung korrekt ist.

Die kritische Beurteilung der Anwendbarkeit verschiedener Triggerkonzepte zur Simulation des SQL-Integritätskonzepts kann in drei Aussagen zusammengefaßt werden.

- Der hier entwickelte triggerbasierte Implementierungsansatz und die Präprozessor-Lösung basierend auf DB-Prozeduren und einem Präprozessor-Schema als Erweiterung des DBS-internen Schemas zeigen, daß die SQL3-Konzepte hinsichtlich ihrer Fähigkeit, SQL3 funktional zu erweitern, sehr leistungsfähig sind, denn eine Erweiterung ist ohne Beschränkungen möglich.

- Obwohl die Integritätsprüfung als ein klassisches Anwendungsgebiet für Trigger gilt, treten bei der Simulation der SQL-Integritätsprüfung mittels SQL-Triggern zwei Probleme auf. Da zum Transaktionsende keine SQL-Trigger ausgeführt werden, ist die triggerbasierte Simulation der DEFERRED-Prüfung sehr aufwendig und nur wenig intuitiv realisierbar. Zudem kann für triggerausgeführte Änderungsanweisungen eine IMMEDIATE-Integritätsprüfung nur durchgeführt werden, wenn je Trigger maximal eine Änderungsanweisung auftritt. Beide Probleme sind jedoch ohne funktionale Beschränkungen lösbar.

- Obwohl die Verfahren die Idee einer auf Triggern basierenden Realisierung von Ceri/Widom ([CW90], [CW91]) und Griefahn ([Grie97]) aufgreifen, unterscheiden sich die Lösungen doch gravierend. Die sehr unterschiedlichen Ausführungsmodelle der Starburst-, ActLog- und SQL-Trigger erfordern sehr verschiedene Ausführungssteuerungen und verschiedene Definitionen der Prozeßtrigger. Decker diskutiert in [Dec01] ein triggerbasiertes IMMEDIATE-Integritätsprüfungsverfahren ohne Änderungspropagierung. Ceri/Fraternali haben in [CF97] für ihre DB2-Trigger-basierte Realisierung eine prozedurale Ausführungssteuerung des DEFERRED-Prozesses gewählt und verzichten somit auf die leistungsfähige Ausführungs- und Relevanzkontrolle durch die Triggerkomponente.

Perspektiven

Das SQL-basierte Verfahren zur inkrementellen Sichtenaktualisierung und Integritätsprüfung wird derzeit im Rahmen verschiedener Diplomarbeiten realisiert. Eine Diplomarbeit beschäftigt sich mit der Implementierung des SQL-Parsers und der inkrementellen Integritätsprüfung. Die zweite setzt sich mit der Simulation materialisierter Sichten sowie der Implementierung der Komponente zur Änderungspropagierung und Sichtenaktualisierung auseinander. Beide Arbeiten werden auf einem DB2 V7.x DB-System von IBM realisiert, so daß der hier vorgestellte triggerbasierte Implementierungsansatz unmittelbar umgesetzt werden kann, denn bei dem DB2-System ist das SQL-Triggerkonzept unverändert realisiert.

Die in Abschnitt 4.3 im Kontext von Datalog und Relationaler Algebra vorgestellten inkrementellen Verfahren für DB-Systeme mit sehr speziellen Eigenschaften können ohne wesentliche Einschränkungen in die SQL-basierten Verfahren integriert werden. Die Definitionen der Prozeßtrigger müssen angepaßt werden, nicht aber die SQL-spezifische Prozeßausführung zu den beiden Zeitpunkten IMMEDIATE und DEFERRED. Zu den Erweiterungsmöglichkeiten zählen u.a. die Zulässigkeit von Duplikaten, Aggregatfunktionen, dynamischen Integritätsbedingungen, Integritätserhaltung und verschiedene Zeitpunkte der Sichtenaktualisierung. Die Berücksichtigung von Duplikaten sowie die inkrementelle Aktualisierung von Aggregatfunktionen würde die Einsetzbarkeit des hier entwickelten Verfahrens für praxisrelevante Anwendungen beachtlich steigern. Für die Anwendbarkeit dynamischer Integritätsbedingungen muß die Syntax von ASSERTIONs um die Möglichkeit zur Formulierung transitionaler Zusammenhänge erweitert werden, was zugleich eine konzeptionelle Erweiterung von SQL3 darstellen würde. Eine Erweiterung um die in Data Warehouse-Anwendungen viel diskutierten Aktualisierungszeitpunkte ON DEMAND und 'periodisch' kann ebenfalls mit nur geringen Anpassungen vorgenommen werden. Zu diesem Zweck dürfen zum Transaktionsende nicht alle Δ-Fakten gelöscht werden. Zur Identifizierung der Δ-Fakten für die nachfolgende Sichtenaktualisierung kann die hier bereits eingeführte Propagierungsprozeßnummer (*pp_id*) zum Ende des DEFERRED-Prozesses mit einer Transaktions-ID belegt werden.

Eine ebenfalls sehr praxisrelevante Erweiterung wäre eine konzeptionelle Erweiterung von SQL3 um die Möglichkeit zur Korrektur eines Integritätsfehlers ('integrity repair') auch für ASSERTIONs. In SQL ist dies bislang nur für Fremdschlüsselbedingungen (FOREIGN KEYs) möglich, indem auf eine Konsistenzverletzung mit dem Löschen weiterer Fakten oder dem Ersetzen fehlerhafter Attributwerte durch einen Standardwert ('default value') oder durch 'NULL' reagiert wird. Die SQL-basierten inkrementelle Verfahren bieten sich wiederum als Basisverfahren an, das ungeachtet der Fehlerreaktion die SQL-spezifischen Anforderungen erfüllt. Eine Lösung wie z.B. in [CW90], wo während der Generierung eines IP-Triggers der Entwickler Korrekturanweisungen programmieren kann, ist ohne viel Aufwand in die gegebene Präprozessor-Lösung integrierbar. Anders als bei Ceri/Widom können hier jedoch Integritätsbedingungen auch für Sichten formuliert werden. Somit ist eine ausführliche Diskussion der Semantik der Korrekturanweisungen erforderlich. Dies gilt insbesondere für Korrekturanweisungen, die Relationen eines niedrigeren und damit bereits bearbeiteten Stratums betreffen. Vor diesem Hintergrund bietet es sich auch an, Änderungsanweisungen auf Sichten ausführen zu können ('view updating'). Für sichtenbasierte Integritätsbedingungen ist es vielfach intuitiver, korrigierende

Änderungsanweisungen unmittelbar für die betroffenen Sichten zu formulieren. Die bekannten Lösungen für die Probleme der Ausführung von Änderungen auf Sichten sind im SQL-Kontext zu diskutieren.

Naheliegend ist zudem die Diskussion einer funktionalen Erweiterung der SQL-Trigger um Trigger, die für Sichten definierbar sind. Die Ableitung induzierter Änderungen könnte dann ein feuerndes Ereignis für sichtenbasierte Trigger darstellen. Eine solche Erweiterung würde insbesondere für das hier entwickelte Verfahren bedeuten, daß die Δ-Fakten nicht zwingend in Basisrelationen gespeichert werden müßten. Unter Berücksichtigung der Abhängigkeiten in einem gegebenen DB-Schema kann dann mit der Zielsetzung effizienterer Laufzeiten individuell für jede Δ-Sicht entschieden werden, ob sie materialisiert oder virtuell vorliegen soll.

Grundsätzlich stellen die Präprozessor-Lösung und das triggerbasierte Verfahren ein geeignetes Vorgehen dar, bestehende DB-Systeme um die fehlende Funktionalität zu erweitern. Die Effizienz der Lösung hängt jedoch unmittelbar von der Effizienz der Triggerausführungen ab. Die Steuerungsaufgaben bei einer triggerbasierten und einer deduktiven Lösung sind die gleichen. Entweder werden die Aufgaben von der Triggerkomponente übernommen, oder aber die Steuerung der Ausführung und die Kontrolle der Relevanz der angewendeten Δ-Sichten werden explizit programmiert. Die Effizienz einer prozeduralen Steuerung wiederum hängt davon ab, ob die Programme im DB-Kern integriert werden können, oder ob Anwendungsprozeduren über Schnittstellen wie 'dynamic SQL' zur Laufzeit die Sichten einbinden und anwenden. Ob eine prozedurale Steuerung schneller ist als der hier vorgeschlagene triggerbasierte Ansatz kann somit nur empirisch evaluiert werden.

Literaturverzeichnis

[Abi88] Abiteboul, S.: *Updates, A New Frontier.* Proc. IDBT 1988, Brügge, Belgien, LNCS 326, S. 1-18.

[AV85] Abiteboul, S., Vianu, V.: *Transactions and Integrity Constraints.* Proc. PODS 1985, Portland/OR, USA, S. 193-204.

[AV87] Abiteboul, S., Vianu, V.: *A Transaction Language Complete for Database Update and Specification.* Proc. PODS 1987, San Diego/CA, USA, S. 260-268.

[AIP96] Asirelli, P., Inverardi, P., Plagenza, G.: *Integrity Constraints as Views in Deductive Databases.* Proc. FMLDO 1996, Schloß Dagstuhl, Deutschland, Inst. für Informatik, Universität Magdeburg, Preprint Nr. 41996, S. 133-140.

[AH88] Abiteboul, S., Hull, R.: *Data Functions, Datalog and Negation.* Proc. SIGMOD 1988, Chicago/IL, USA, S. 143-153.

[ANSI99I] ANSI/ISO/IEC 9075-1-1999: *Database Languages - SQL - Part 1: Framework.* ANSI American National Standards Institute, New York, 1999. http://www.cssinfo.com/ncitsgate.html.

[ANSI99II] ANSI/ISO/IEC 9075-2-1999: *Database Languages - SQL - Part 2: Foundation.* ANSI American National Standards Institute, New York, 1999. http://www.cssinfo.com/ncitsgate.html.

[Bid91] Bidoit, N.: *Negation in Rule-Based Database Languages: A Survey.* Theoretical Computer Science, TSC 78(1), Elsevier Science Publishers, North-Holland, S. 3-83, 1991.

[BB82] Bernstein, P.A., Blaustein, B.T.: *Fast Methods for Testing Quantified Relational Calculus Assertions.* Proc. SIGMOD 1982, Orlando/FL, USA, S. 39-50.

[BBC80] Bernstein, P.A., Blaustein, B.T., Clarke, E.M.: *Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data.* Proc. VLDB 1980, Montreal, Kanada, S. 126-136.

[BBKL+88] Barthez, M., Borla, P., Kiernan, J., Lefebvre, N., de Maindreville, C., Portier, M.A., de Sahb, N., Simon, E.: *An Overview of the RDL1 Deductive Database System.* Proc. Annual ESPRIT Conference 1988, Brüssel, Belgien, S. 845-862.

[BDDF+98] Bello, R.G., Dias, K., Downing, A., Feenan, J., Finnetry, J., Norcott, W.D., Sun, H., Witkowski, A., Ziauddin, M.: *Materialized Views in Oracle.* Proc. VLDB 1998, New York, USA, S. 659-664.

[BDM88] Bry, F., Decker, H., Manthey, R.: *A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases.* Proc. EDBT 1988, Venedig, Italien, LNCS 303, S. 488-505.

[BK95] Bonner, A.J., Kifer, M.: *Transaction Logic Programming (or A Logic of Declarative and Procedural Knowledge).* Technical Report CSRI-323, University of Toronto, Kanada, 1995.
http://www.cs.toronto.edu/ bonner/transaction-logic.html.

[BK98] Bonner, A.J., Kifer, M.: *The State of Change: A Survey.* Proc. Int. Workshop on Logic Databases and the Meaning of Change, 1996, Schloß Dagstuhl, Deutschland, Transactions and Change in Logic Databases, LNCS 1472, S. 1-36, 1998.

[BMM92] Bry, F., Manthey, R. Martens, B.: *Integrity Verification in Knowledge Bases.* Proc. Russian Conf. on Logic Programming 1991, St. Petersburg, Rußland, LNCS 592, S. 114-139.

[BMP01] Behrend, A., Manthey, R. Pieper, B.: *An Amateur's Introduction to Integrity Constraints and Integrity Checking in SQL.* Proc. BTW 2001, Oldenburg, Deutschland, Informatik Aktuell, Springer, Berlin, Deutschland, S. 405-423.

[CCW00] Ceri, S., Cochrane R.J., Widom, J.: *Practical Applications of Triggers and Constraints: Successes and Lingering Issues.* Proc. VLDB 2000, Kairo, Ägypten, S. 254-262.

[CD83] Cremers, A.B., Domann, G.: *AIM - An Integrity Monitor for the Database System INGRES.* Proc. VLDB 1983, Florenz, Italien, S. 167-170.

[Cel00] Celko, J.: *SQL for Smarties: Advanced SQL Programming.* Morgan Kaufmann, San Franzisco/CA, USA, 2000.

[CF97] Ceri, S., Fraternali, P.: *Designing Database Applications with Objects and Rules - The IDEA Methodology.* Addison Wesley, Großbritanien, 1997.

[CFPT94] Ceri, S., Fraternali, P., Paraboschi S., Tanca L.: *Automatic Generation of Production Rules for Integrity Maintenance.* ACM TODS 19(3), S. 367-422, 1994.

[CG85] Ceri, S., Gottlob, G.: *Translating SQL Into Relational Algebra: Optimization, Semantics and Equivalence of SQL Queries.* IEEE Transactions on Software Engineering SE-11(4), S. 324-345, 1985.

[CGH94] Cremers, A.B., Griefahn, U., Hinze, R.: *Deduktive Datenbanken - Eine Einführung aus der Sicht der logischen Programmierung.* Vieweg Verlag, Braunschweig, Deutschland, 1994.

[CGLMT96] Colby, L.S., Griffin, T., Libkin, L., Mumick, I.S., Trickey, H.: *Algorithms for Deferred View Maintenance.* Proc. SIGMOD 1996, Montreal, Kanada, S. 469-480.

[CGT90] Ceri, S., Gottlob, G., Tanca, L.: *What You Always Wanted to Know About Datalog (And Never Dared to Ask)* . IEEE TKDE 1(1), S. 146-166, 1989.

[CGT90] Ceri, S., Gottlob, G., Tanca, L.: *Logic Programming and Databases.* Surveys in Computer Science, Springer-Verlag, Berlin, Deutschland, 1990.

[ChT95] Chomicki, J., Toman, D.: *Implementing Temporal Integrity Constraints Using an Active DBMS.* IEEE TKDE 7(4), S. 566-582, 1995. http://www.cis.ksu.edu/ chomicki/journal.html.

[CKLMR97] Colby, L.S., Kawaguchi, A., Lieuwen, D.F., Mumick, I.S., Ross, K.A.: *Supporting Multiple View Maintenance Policies.* Proc. SIGMOD 1997, Tuscon/AZ, USA, S. 405-416.

[Codd86] Codd, E.F.: *Missing Information (Applicable and Inapplicable) in Relational Databases.* SIGMOD Record 15(4), S. 53-78, 1986.

[Codd87] Codd, E.F.: *More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable Information).* SIGMOD Record 16(1), S. 42-50, 1987.

[Codd88] Codd, E.F.: *Fatal Flaws in SQL.* Datamation, S. 45-48, 1988.

[CPM96] Cochrane R.J., Pirahesh, H., Mattos, N.: *Integrating Triggers and Declarative Constraints in SQL Database Systems.* Proc. 22. VLDB, Mumbai, Indien, S. 567-578, 1996.

[Cre92] Cremers, A.B.: *Informationssysteme.* Skript zur Vorlesung; Inst. für Informatik, Universität Bonn, Deutschland, Sommersemester 1992.

[CW90] Ceri, S., Widom, J.: *Deriving Production Rules for Incremental Constraint Maintenance.* Proc. VLDB 1990, Brisbane, Australien, S. 566-577.

[CW91] Ceri, S., Widom, J.: *Deriving Production Rules for Incremental View Maintenance.* Proc. VLDB 1991, Barcelona, Spanien, S. 577-589.

[CW94] Ceri, S., Widom, J.: *Deriving Incremental Production Rules for Deductive Data.* Information Systems 19(6), Elsevier Science, S. 467-490, 1994.

[Das92] Das, S.K.: *Deductive Databases and Logic Programming.* Addison-Wesley, Großbritanien, 1992.

[Date86] Date, C.J.: *Relational Database - Selected Writings 1986.* Addison-Wesley, USA, 1986.

[Date94] Date, C.J.: *An Introduction to Database Systems.* 6. Edition, Addison-Wesley, USA, 1994.

[DD97] Date, C.J., Darwen, H.: *A Guide to the SQL Standard.* 4. Edition, Addison-Wesley, USA, 1997.

[Dec86] Decker, H.: *Integrity Enforcement on Deductive Databases.* Proc. Int. Conf. on Expert Database Systems 1986, Charleston/SC, USA, S. 381-395.

[Dec01] Decker, H.: *Soundcheck for SQL.* Proc. Practical Aspects of Declarative Languages, PADL 2001, Las Vegas, Nevada, S. 214-228,.

[DG96] Dittrich, K.R., Gatziu, S.: *Aktive Datenbanksysteme - Konzepte und Mechanismen.* International Thomson Publishing, Bonn, Deutschland, 1996.

[DS00] Dong, G., Su, J.: *Incremental Maintenance of Recursive Views Using Relational Calculus/SQL.* SIGMOD Record 29(1), S. 44-51, 2000.

[DW89] Das, S.K., Williams, M.H.: *A Path Finding Method for Constraint Checking in Deductive Databases.* Data and Knowledge Engineering 4, Elsevier Science Publisher, S. 223-244, 1989.

[Elkan90] Elkan, C.: *Independence of Logic Queries and Updates.* Proc. PODS 1990, Nashville, Tennessee, S. 154-160.

[EM00] Eisenberg, A., Melton, J.: *SQL Standardization: The Next Steps.* SIGMOD Record 29(1), S. 63-67, 2000.

[GaMi78] Gallaire, H., Minker, J. (Hrsg.): *Logic and Databases.* Plenum Press, New York, USA, 1978.

[Ger96] Gertz, M.: *Diagnosis and Repair of Constraint Violations in Database Systems.* Dissertation, Fachbereich Mathematik, Universität Hanover, Deutschland, DISDBIS Bd. 19, infix Verlag, Sankt Augustin, Deutschland, 1996.

[GL95] Griffin, T., Libkin, L.: *Incremental Maintenance of Views With Duplicates.* Proc. SIGMOD 1995, San José/CA, USA, S. 328-339.

[GMS93] Gupta, A., Mumick, I.S., Subrahmanian, V.S.: *Maintaining Views Incrementally.* Proc. SIGMOD 1993, Washington/DC, USA, S. 157-166.

[GSUW94] Gupta, A., Sagiv, Y., Ullman, J.D., Widom, J.: *Constraint Checking with Partial Information.* Proc. SIGMOD/PODS 1994, Minneapolis/MI, USA, S. 45-55.

[GP99] Gulutzan, P., Pelzer, T.: *SQL-99 Complete, Really.* R&D Books Miller Freeman, Lawrence/KS, USA, 1999.

[Grie97] Griefahn, U.: *Reactive Model Computation - A Uniform Approach to the Implementation of Deductive Databases.* Dissertation; Institut für Informatik III, Rheinische Friedrich-Wilhelms-Universität, Bonn, Deutschland, 1997, http://www.informatik.uni-bonn.de/~ulrike/Publications/publications.html.

[GrMa94] Griefahn, U., Manthey, R.: *Update Propagation in Chimera, an Active DOOD Language.* Proc. DAISD 1994, Aiguablara, Spanien, Report de recerca LSI/94-28-R, Universität Barcelona, Spanien, S. 277-298.

[GrMi89] Grant, J., Minker, J.: *Deductive Database Theories.* The Knowledge Engineering Review 4(4), 267-304, 1989.

[GuMu95] Gupta, A., Mumick, I.S.: *Maintenance of Materialized Views: Problems, Techniques and Applications.* IEEE Data Engineering Bulletin 18(2), S. 3-18, 1995.

[GW99] Groff, J.R., Weinberg, P.N.: *SQL: The Complete Reference.* Osborne/Mc Graw Hill, Berkley/CA, USA, 1999.

[HS95] Heuer, A., Saake, G.: *Datenbanken - Konzepte und Sprachen.* International Thomson Publishing, Bonn, Deutschland, 1995.

[HW93] Hanson, E.N., Widom, J.: *An Overview of Production Rules in Database Systems.* The Knowledge Engineering Review 8(2), Cambridge University Press, S. 121-143, 1993.

[HZ96] Hull, R., Zhou, G.: *A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches.* Proc. SIGMOD 1996, Montreal, Canada, S. 481-492.

[IBM00] IBM: *SQL-Referenz DB2 V7.* IBM Corp., New York, USA, SC09-2974-00, SC09-2975-00; 2000;
ftp://ftp.software.ibm.com/ps/products/db2/info/vr7/

[KdMS90] Kiernan, J., de Maindreville, C., Simon, E.: *Making Deductive Databases a Practical Theory: A Step Forward.* Proc. SIGMOD 1990, Atlantic City/NY, USA, S. 237-246.

[KE97] Kemper, A., Eickler, A.: *Datenbanksysteme - Eine Einführung.* Oldenbourg Verlag, München, Deutschland, 1997.

[KK93] Kandzia, P., Klein, H.-J.: *Theoretische Grundlagen relationaler Datenbanksysteme.* Reihe Informatik Bd 79, BI Wissenschaftsverlag, Mannheim, Deutschland, 1993.

[Klein94] Klein, H.-J.: *How to Modify SQL Queries in Order to Guarantee Sure Answers.* SIGMOD Record 23(3), S. 14-20, 1994.

[Klein97] Klein, H.-J.: *Gesicherte und mögliche Antworten auf Anfragen an relationale Datenbanken mit partiellen Relationen.* Habilitation, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität, Kiel, Deutschland, 1997.

[Kue91] Küchenhoff, V.: *On the Efficient Computation of the Differences Between Consecutive Database States.* Proc. DOOD 1991, München, Deutschland, LNCS 566, S. 478-502.

[KR99] Kotidis, Y., Roussopoulos, N.: *DynaMat: A Dynamic View Management System for Data Warehouses.* Proc. SIGMOD 1999, Philadelphia/PA, USA, S. 371-382.

[KSS87] Kowalski, R., Sadri, F., Sopor, P.: *Integrity Checking in Deductive Databases.* Proc. VLDB 1987, Brighton, Großbritannien, S. 61-69.

[Lip88] Lipeck, U.W.: *Dynamische Integrität von Datenbanken.* Informatik-Fachberichte 209, Springer Verlag, Berlin, Deutschland, 1988.

[LCCR94] Laender, A.H.F., Casanova, M.A., de Carvalho, A.P., Ridolfi, L.F.G.G.M.: *An Analysis of SQL Integrity Constraints from an Entity-Relationship Model Perspective.* Information Systems 19(4), S. 331-358, 1994.

[LL96] Lee, S.Y., Ling, T.W.: *Further Improvement on Integrity Constraint Checking for Stratifiable Deductive Databases.* Proc. VLDB 1996, Mumbai, Indien, S. 495-505.

[LMR96] Ludäscher, B., May, W., Reinert, J.: *Towards a Logical Semantics for Referential Actions in SQL.* Proc. Int. Workshop on Foundations of Models and Languages for Data and Objects, FMLDO 1996, Schloß Dagstuhl, Deutschland, S. 57-72.

[LMSS95] Lu, J.J., Moerkotte, G., Schue, J., Subrahmanian, V.S.: *Efficient Maintenance of Materialized Mediated Views.* Proc. SIGMOD 1995, San José/CA, USA, S. 340-351.

[LS93] Levy A.Y., Sagiv, Y.: *Queries Independent of Updates.* Proc. VLDB 1993, Dublin, Irland, S. 171-181.

[LST86] Lloyd, J.W., Sonenberg, E.A., Topor, R.W.: *Integrity Constraint Checking in Stratified Databases.* Technical Report 86/5, Department of Computer Science, University of Melbourne, Australien, 1986.

[Man94] Manthey, R.: *Reflections on Some Fundamental Issues of Rule-Based Incremental Update Propagation.* Proc. DAISD 1994, Aiguablara, Spanien, Report de recerca LSI/94-28-R, Universität Barcelona, Spanien, S. 255-276.

[ML91] Moerkotte, G., Lockemann, P.C.: *Reactive Consistency Control in Deductive Databases.* TODS 16(4), S. 670-702, 1991.

[MS93] Melton, J., Simon, A.R.: *Understanding the New SQL: A Complete Guide.* Morgan Kaufmann, San Franzisco/CA, USA, 1993.

[MT95] Mayol, E., Teniente, E.: *Towards an Efficient Method for Updating Consistent Deductive Databases.* Proc. Basque Int. Workshop on Information Technology, BIWIT 1995, San Sebastian, Spanien, IEEE Computer Society Press, S. 113-123.

[MT99] Mayol, E., Teniente, E.: *Adressing Efficiency Issues During the Process of Integrity Maintenance.* Proc. DEXA 1999, Florenz, Italien, LNCS 1677, S. 270-281.

[MT00] Mayol, E., Teniente, E.: *Dealing with Modification Requests During View Updating and Integrity Constraint Maintenance.* Proc. FoIKS 2000, Burg, Deutschland, 1762 LNCS, S. 192-212.

[MW88] Manchanda, S., Warren, D.S.: *A Logic-based Language for Database Updates.* in J. Minker (Hrsg.): Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, San Franzisco/CA, USA, S. 363-394, 1988.

[Neu94] Neumann, K.: *Formulierung von Integritätsbedingungen in verschiedenen SQL-Dialekten.* Proc. Workshop Aktive Datenbanken 1994, GI-Rundbrief, 14, S. 17-21.

[NM97] Neumann, K., Müller, R.: *Implementierung von Assertions durch Oracle7-Trigger.* Informatik Berichte 97(02), Technische Universität Braunschweig, S. 1-16, 1997.

[Nic82] Nicolas, J.-M.: *Logic for Improving Integrity Checking in Relational Data Bases.* Acta Informatica 18(3), S. 227-253, 1982.

[NK88] Naqvi, S., Krishnamurthy, R.: *Database Updates in Logic Progamming.* Computer Science Press, 1988.

[NPS91] Negri, M., Pelagatti, G., Sbatella, L.: *Formal Semantics of SQL Queries.* TODS 17(3), S. 513-534, 1991.

[NT89] Naqvi, S., Tsur, S. (Hrsg.): *A Logical Language for Data and Knowledge Bases.* Proc. SIGMOD-SIGACT-SIGART 1989, Austin/TX, USA, S. 251-262.

[Oli91] Olivé, A.: *Integrity Constraints Checking in Deductive Databases.* Proc. VLDB 1991, Barcelona, Spanien, S. 513-523.

[Ora99] Lorentz, D., et.al.: *Oracle8i SQL Reference (8.1.6).* Oracle Corporation, Redwood City, USA, Part No. A76989-01, 1999.

[Pat99] Paton N.W. (Hrsg.): *Active Rules in Database Systems.* Springer Verlag, Berlin, Deutschland, 1999.

[Pie00] Pieper, B.: *Inkrementelle Integritätsprüfung und Anpassung materialisierter Sichten in SQL.* Proc. Workshop Grundlagen von Datenbanken 2000, Plön, Deutschland, Bericht Nr. 2005, Universität Kiel, Deutschland, S. 71-75.

[QS87] Qian, X., Smith, D.R.: *Integrity Constraint Reformulation for Efficient Validation.* Proc. VLDB 1987, Brighton, Großbritannien, S. 417-425.

[QW91] Qian, X., Wiederhold, G.: *Incremental Recomputation of Active Relational Expressions* . IEEE TKDE 3(3), S. 337-341, 1991.

[Rei96] Reinert, J.: *Ein Regelsystem zur Integritätssicherung in aktiven relationalen Datenbanksystemen.* Dissertation, Universität Kaiserslautern, Deutschland, DISDBIS Bd. 11, infix Verlag, Sankt Augustin, Deutschland, 1996.

[RSS96] Ross, K.A., Srivastava, D., Sudarshan, S.: *Materialized View Maintenance and Integrity Constraint Checking: Trade Space for Time.* Proc. SIGMOD 1996, Montreal, Kanada, S. 447-458.

[SA95] Schlesinger, M., Ackermann, F.: *Vergleichende Gegenüberstellung der Trigger-Mechanismen von Ingres, Oracle und Sybase.* in H.-J. Scheibt (Hrsg.): Softwareentwicklung - Methoden, Werkzeuge, Erfahrungen, Technische Akademie Esslingen, S. 41-53, 1995.

[SBLC00] Salem, K., Beyer, K., Lindsay, B., Cochrane, R.: *How To Roll a Join: Asynchronous Incremental View Maintenance.* Proc. SIGMOD 2000, Dallas/TX, USA, S. 129-140.

[Schoe95] Schöning, U.: *Logik für Informatiker.* Spektrum Akademischer Verlag, Heidelberg, Deutschland, 1995.

[Spr95] Spruit, P.A.: *Logics of Database Updates.* Dissertation, Universität Amsterdam, Niederlande, 1994.

[ST96] Schewe, K.-D., Thalheim, B.: *Active Consistency Enforcement for Repairable Database Transitions.* Proc. Int. Workshop on Foundations of Models and Languages for Data and Objects, FMLDO 1996, Schloß Dagstuhl, Deutschland, S. 87-102.

[TO95] Teniente, E., Olivé, A.: *Updating Knowledge Bases While Maintaining Their Consistency.* VLDB Journal 4, S. 193-241, 1995.

[UO92] Urpí, T., Olivé, A.: *A Method for Change Computation in Deductive Databases.* Proc. VLDB 1992, Vancouver, Kanada, S. 225-237.

[UO94] Urpí, T., Olivé, A.: *Semantic Change Computation Optimization in Active Databases.* Proc. RIDE-ADS 1994, Houston/TX, USA, S. 19-27.

[Ull89] Ullman, J. D.: *Principles of Database and Knowledge-Base Systems.* Volume I and II, Computer Science Press, Rockville/MA, USA, 1989, 1990.

[vEK76] van Emden, M.,H., Kowalski, R.,A.: *The Semantics of Predicate Logic as Progamming Language.* Journal of the ACM 23(4), S. 733-742, 1976.

[WC96] Widom, J., Ceri, S. (Hrsg.): *Active Database Systems - Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, San Franzisco/CA, USA, 1996.

[WF90] Widom, J., Finkelstein, S.J.: *Set-Oriented Production Rules in Relational Database Systems.* Proc. SIGMOD 1990, Atlantic City/NY, USA, S. 259-270.

[WFF98] Wichert, C.-A., Freitag, B., Fent, A.: *Logical Transactions and Serializability.* Proc. Int. Workshop on Logic Databases and the Meaning of Change, Schloß Dagstuhl, Deutschland, Transactions and Change in Logic Databases, LNCS 1472, S. 134-165, 1998.

[Wid91] Widom, J.: *Deduction in the Starburst Production Rule System.* Research Report, RJ 8135 (74548) Computer Science, IBM Almanden Research Center, San José/CA, USA, S. 1-21, 1991.

[Wid93] Widom, J.: *Deductive and Active Databases: Two Paradigms or Ends of a Spectrum?.* Proc. RIDS 1993, Edinburgh, Scotland, S. 306-315.

[Wid96] Widom, J.: *The Starburst Active Database Rule System.* IEEE TKDE 8(4), S. 583-595, 1996.
http://www-db.stanford.edu/pub/widom.

Abbildungsverzeichnis

Curriculum Vitae

Birgit Pieper
In der Taufe 6
51427 Bergisch-Gladbach

Geboren am 13. Januar 1966 in Hamm
Familienstand: verheiratet

1972 - 1976	Carl-Orff Grundschule in Hamm
1976 - 1982	Realschule des St. Ursula-Stiftes in Werl
1982 - 1985	Gymnasium des St. Ursula-Stiftes in Werl
Mai 1985	Abitur
1985 - 1988	Ausbildung zur Datenverarbeitungskauffrau bei der FINK-Chemie GmbH in Hamm
1988 - 1991	Studium der Wirtschaftsinformatik an der Fachhochschule des Landes Rheinland-Pfalz in Worms
September 1991	Einreichung der Diplomarbeit *"Entwurf eines Anwendungssystems für die Verwaltung und Bearbeitung aller Studentendaten an der Fachhochschule Worms und die Realisierung des Teilsystems Stammdatenverwaltung unter Verwendung des DBS INGRES"* Abschluß: Diplom-Informatikerin (FH)
1991 - 1996	Studium der Informatik mit Nebenfach Rechtsinformatik an der Rheinischen Friedrich-Wilhelms-Universität in Bonn
Januar 1996	Einreichung der Diplomarbeit *"Inkrementelle Integritätsprüfung mittels aktiver Regeln in Phoenix"* Abschluß: Diplom-Informatikerin
seit März 1996	Angestellte bei der Opitz Consulting GmbH in Gummersbach