

# Die SQL-Normen\*

## Normen der Reihe ISO/IEC 9075

Die Internationale Norm ISO/IEC 9075 „Informationstechnik – Datenbanksprachen – SQL“ ist seit den späten achtziger Jahren des zwanzigsten Jahrhunderts eine der wichtigsten Grundlagennormen für Datenbankanwendungen weltweit. Im Dezember 2003 veröffentlichten ISO und IEC die neueste Ausgabe der ISO/IEC „Structured Query Language“, so die Langform für „SQL“ – auf deutsch „strukturierte Abfragesprache“.

Seit ihrem ersten Erscheinen im Jahr 1987 wird ISO/IEC 9075 im Arbeitsausschuss JTC 1/SC 32 „Datenmanagement und Datenaustausch“ des Internationalen Komitees ISO/IEC JTC 1 „Informationstechnik“ kontinuierlich weiterentwickelt. Experten des deutschen Spiegelgremiums NI-32 im Normenausschuss Informationstechnik (NI) sind an der Normungsarbeit auf diesem Gebiet aktiv beteiligt. Allein in den Jahren 2000 bis 2003 haben Mitarbeiter des NI-32 in 34 Beiträgen mehr als 120 Änderungsvorschläge eingebracht.

In diesem Artikel geben die Autoren einen Überblick über die neuesten Funktionen der ISO/IEC 9075 SQL-Norm, die im Artikel der Einfachheit halber als SQL:2003 bezeichnet wird. Dabei beginnen sie mit einer kurzen Zusammenfassung der vorhandenen Funktionalität in der bis Ende 2003 gültigen ISO/IEC 9075 (SQL:1999) und den früheren Versionen. Zusätzlich wird auf ausgewählte Teile der Norm ausführlicher eingegangen und abschließend ein kurzer Ausblick auf die Zukunft der SQL-Norm aus Sicht der Autoren gegeben.

### 1 Einleitung

Die Datenbanksprache SQL hat sich heute fest etabliert als die bestimmende Norm für den Zugriff auf Datenbanken. Man kann ohne

\*) Dieser Beitrag ist im Original in „it – Information Technology“, 45 (2003) 1, Seite 30 bis 38 erschienen. Wir drucken die Übersetzung mit freundlicher Genehmigung des Oldenbourg Verlags.

Die Verfasser des Originalartikels sind *Jan-Eike Michels, Krishna Kulkarni, Christopher M. Farrar, Andrew Eisenberg, Nelson Mattos, Hugh Darwen, IBM Silicon Valley Lab, San José, USA.*

Die im Original englischsprachige Fassung des Artikels wurde von den Normungsexperten des NI-32 *Andreas Osterhold* und *Peter Pistor* (Obmann NI-32) übersetzt und in einigen Punkten aktualisiert.



Andreas Osterhold



Peter Pistor

Übertreibung sagen, dass fast jeder, der im IT-Bereich tätig ist, in irgendeiner Weise mit SQL vertraut ist. Es ist auch keine Übertreibung, dass die Veröffentlichung von SQL als Internationale Norm der Reihe ISO/IEC 9075 in den späten 80ern des letzten Jahrhunderts entscheidend verantwortlich war für den Aufschwung einer Multi-Milliarden-Dollar-Industrie, deren Angebot Datenbanken, Entwicklungswerkzeuge und eine breit gefächerte Applikationssoftware umfasst.

Die Normungsaktivitäten für SQL – sowohl von Systemanbietern als auch von Endbenutzern vorangetrieben – gibt es seit etwa zwei Jahrzehnten. Die erste Version der SQL-Norm wurde von ANSI 1986 veröffentlicht. Eine identische Version erschien 1987 als Internationale Norm der ISO unter der Mitwirkung vieler nationaler Normungsorganisationen. In der Folge gab es drei Revisionen der Norm. Die erste erschien 1989, die zweite 1992 und die dritte 1999. Eine weitere Revision ist mit Ausgabedatum Dezember 2003 erschienen.

Im nächsten Kapitel geben wir einen Überblick über die bisherigen Versionen der SQL-Norm einschließlich der bis Ende 2003 aktu-

ellen Version (SQL:1999). Es folgt eine kurze Beschreibung ausgewählter Funktionen, die in der nächsten Revision der Norm enthalten sein werden. Danach findet der Leser drei Abschnitte mit kurzen Beschreibungen einiger Teile des SQL:2003-Normenpaketes. Eine kurze Beschreibung der Zukunft von SQL schließt den Artikel ab. Aus Platzgründen kann hier nur die Oberfläche eines jeden Themas behandelt werden. Interessierte Leser verweisen wir auf die Veröffentlichungen in den Referenzen am Ende des Artikels. Details über SQL:1999 können besonders in [1] und [2] gefunden werden. Die Internationale Norm kann direkt bei ISO ([www.iso.ch](http://www.iso.ch)) oder im Beuth Verlag ([www.beuth.de](http://www.beuth.de)) gefunden werden.

### 2 Von SQL:1986 bis SQL:1999

#### SQL:1986 und SQL:1989

Wie bereits erwähnt, wurde die erste Version der SQL-Norm 1986 veröffentlicht. SQL war nicht als vollständige Programmiersprache entworfen worden, sondern als Anfragesprache. Um Anwendungen zu entwickeln, mussten SQL-Anweisungen in andere (so genannte berechnungsvollständige) Programmiersprachen (auch *Host-* oder *Wirtsprachen* genannt) wie C, Cobol und andere eingebettet werden. SQL:1986 unterstützt benannte Tabellen, die aus einer Sammlung von Zeilen bestehen (satzähnliche Strukturen, von denen jede eine feste Anzahl Spalten hat). SQL:1986 stellt ebenfalls eine DDL (*data definition language*) mit Konstrukten zum Erzeugen von Tabellen bereit, eine DML (*data manipulation language*) mit Konstrukten zum Einfügen, Aktualisieren und Löschen von Zeilen aus Tabellen und schließlich Konstrukten zur Spezifikation von Query-Ausdrücken, mit denen Zeilen aus einer oder aus mehreren Tabellen erfragt werden können. SQL:1986 bietet eine fest vorgegebene Menge eingebauter Typen, mit deren Hilfe die Datentypen der Werte festgelegt werden, die in den einzelnen Spalten gespeichert werden können (dem entsprechend sind SQL-Ausdrü-

cke streng getypt). SQL:1986 enthält ferner den Begriff der Privilegien, mit denen der Zugriff auf Tabellen kontrolliert werden kann, und grundlegende Möglichkeiten zur Spezifizierung deklarativer Integritätsbedingungen.

1989 wurde eine Revision der Norm SQL:1989 veröffentlicht. Diese Version erweitert SQL:1986 um den Begriff der referentiellen Integrität. Obwohl das eine signifikante Erweiterung war, enthält SQL:1989 keine neuen DDL-, DML- oder *Query*-Konstrukte.

### SQL:1992

Eine wesentliche Revision der Norm erschien 1992 als SQL:1992. Diese brachte Erweiterungen in fast allen Teilen der Sprache. Darunter fallen zusätzliche Datentypen, *Outer joins*, Kataloge, Domänen, Zuweisungen, temporäre Tabellen, referentielle Aktionen, eine *Schema Manipulation Language*, dynamisches SQL, *Information Schema Tables* sowie eine größere Orthogonalität der Sprache selbst.

Kurze Zeit später entschloss man sich, die Weiterentwicklung der Norm in Form inkrementeller Teile zu betreiben. Das heißt, dass von nun an nicht mehr eine komplette monolithische Norm in regelmäßigen Abständen überarbeitet und neu veröffentlicht wird, sondern dass SQL in kleinen Teilen erweitert werden kann und man damit zu vernünftigeren Zeitplänen kommt. Die erste Teil-Norm erschien 1995, ein neues *Call Level Interface* (SQL/CLI). CLI funktioniert als aufrufbare Schnittstelle zu einem SQL-Datenbank-System und stellt so eine hohe Dynamik bereit, ganz im Gegensatz zu den relativ statischen Eigenschaften von eingebundenem SQL. SQL/CLI wird hauptsächlich durch Ad-hoc-Anwendungen genutzt, etwa bei Entscheidungs-Unterstützungs-Systemen, wohingegen eingebettetes (*embedded*) SQL eher bei Applikationen genutzt wird, die weniger dynamisch in ihren Funktionen sind.

Die zweite inkrementelle Teil-Norm, SQL/PSM (*Persistent Stored Modules*), erschien 1996. SQL/PSM standardisiert prozedurale Konstrukte, die genutzt werden, um so genannte *Stored procedures* zu entwickeln. Anders ausgedrückt: Die Norm ermöglicht es, Anwendungs-Logik aus den Anwendungsprogrammen in das Datenbank-System zu verlagern. SQL/PSM unterstützt zusammengesetzte Ausdrücke in Block-Struktur (also Gruppen von SQL-Anweisungen, die als Einheit ausgeführt werden), Verzweigungs-Anweisungen, lokale Variable und Bedingungs-Handler zur Behandlung von Ausnahmefällen. Darüber hinaus unterstützt SQL/PSM aus SQL heraus aufrufbare Routinen (SQL-aufrufbare Routinen, in der Norm *SQL*

*invoked routines* genannt), die entweder Funktionen oder Prozeduren sein können. Funktionen können überladen werden; die Funktion mit der besten Übereinstimmung wird zur Übersetzungszeit ausgewählt.

### SQL:1999

1999 gab es eine weitere Revision der Norm, SQL:1999. Diese erweitert die Idee der inkrementellen Teile und schafft damit eine fünfteilige Norm, die aus folgenden Teilen besteht:

- Teil 1: SQL/Framework
- Teil 2: SQL/Foundation
- Teil 3: SQL/CLI
- Teil 4: SQL/PSM und
- Teil 5: SQL/Bindings.

In Teil 1: SQL/Framework wird die Beziehung zwischen den verschiedenen Teilen beschrieben. Zusätzlich werden Begriffe, Definitionen und *Conformance*-Anweisungen beschrieben, die für alle Teile gelten. Der Teil 2: SQL/Foundation ist der umfangreichste Teil; er enthält alle DDL- und DML-Elemente von SQL:1992 (allerdings ohne die Spezifikationen für *embedded* und *dynamic* SQL), die SQL-aufrufbaren Routinen von SQL/PSM sowie viele neue Sprachmerkmale, die seit der Veröffentlichung von SQL:1992 entwickelt wurden. Für die SQL:1999-Versionen sowohl von Teil 3: SQL/CLI als auch von Teil 4: SQL/PSM wurden die entsprechenden Versionen von 1995 bzw. 1996 geringfügig überarbeitet. Teil 5: SQL/Bindings enthält die Spezifikationen für *embedded* und *dynamic* SQL von SQL-92, entsprechend überarbeitet, wobei die neuen Merkmale aus SQL/Foundation berücksichtigt wurden.

Im Vergleich zu seinem Vorgänger, SQL:1992, bietet Teil 2: SQL/Foundation von SQL:1999 viele Neuerungen. Der wesentliche Schwerpunkt dieser Neuerungen ist die Erweiterung der Sprache um objekt-rationale Elemente. SQL:1999 bietet ein ziemlich reiches Typ-System mit zusätzlichen vordefinierten Datentypen (BOOLEAN, CHARACTER LARGE OBJECT (CLOB) und BINARY LARGE OBJECT (BLOB)), Typ-Konstruktoren (REF, ROW und ARRAY) und der Fähigkeit, benutzerdefinierte Datentypen zu erzeugen.

Der Typ BOOLEAN erlaubt Benutzern, ohne Umwege die Werte *true*, *false* und *unknown* in Spalten abzulegen, wohingegen es die Typen CLOB und BLOB dem Benutzer erlauben, große Binär- und Zeichenketten als Spaltenwerte zu speichern. Der ROW-Typ von SQL:1999 ist eine Erweiterung des anonymen Zeilentyps, den SQL schon immer hatte, der aber für Nutzer nicht explizit verfügbar war, um zum Beispiel einen Spaltentyp zu definieren. Der ROW-Typ-Konstruktor erlaubt Da-

tenbank-Designern, mit strukturierten Werten einzelne Spalten der Datenbank zu bevölkern. Der Array-Typ gestattet es, geordnete Mengen von Werten direkt in einer Spalte einer Datenbanktabelle zu speichern.

Benutzerdefinierte Typen (UDTs) gibt es in zwei Arten, nämlich als ausgeprägte Typen (*distinct types*) und als strukturierte Typen. Ein ausgeprägter Typ ist ein benannter Typ, der auf einem eingebauten Typ basiert (auch Quelltyp genannt). Ein ausgeprägter Typ bietet strenge Typbindung. Er ist gegenüber seinem Quelltyp dadurch „ausgeprägt“ (oder ausgezeichnet), dass ein Vergleich oder eine Zuweisung zwischen Werten des ausgeprägten Typs und seinem Quelltyp grundsätzlich nicht erlaubt ist. Ein strukturierter Typ entspricht einer Menge von Attributen und Methodendefinitionen. Eine Methode ist ähnlich einer Funktion mit einem ausgezeichneten Parameter, der *subject parameter* genannt wird, und dessen Typ der strukturierte Typ ist, für den die Methode definiert wurde. Ein strukturierter Typ ist vollständig verkapselt; das heißt, nur sein Verhalten ist außerhalb der Typdefinition sichtbar, nicht aber die Implementierung der Attribute und Methoden. Insbesondere steht jedes Attribut für ein Paar spezieller Methoden, die den Attributwert verändern und auslesen. Verkapselung ist das, was strukturierte Typen von Zeilentypen unterscheidet. Während ein Zeilentyp durch seine Felder spezifiziert wird, wird ein strukturierter Typ durch seine Methoden spezifiziert.

Darüber hinaus können strukturierte Typen in Subtyp-Supertyp-Beziehungen stehen. Werte von Subtypen kann man überall dort einsetzen, wo Werte eines Supertyps erwartet werden. Die Vorauswahl überladener Methoden findet zur Übersetzungszeit statt und basiert auf einem „uneigennütigen“ (*unselfish*) Algorithmus, bei dem die Typen aller Argumente eines Methodenaufrufs berücksichtigt werden, wohingegen die Endauswahl zur Laufzeit auf einem „eigennütigen“ (*selfish*) Algorithmus basiert, der auf dem Laufzeittyp des Subjekt-Parameters der Methode basiert.

UDTs können überall dort verwendet werden, wo ein Datentyp verwendet werden kann (zum Beispiel als Datentyp einer Spalte). Ein strukturierter Typ lässt sich aber auch auf eine ganz neue Art einsetzen. Er kann mit einer Tabelle (oder einer Sicht) assoziiert werden, typisierte Tabelle (oder typisierte Sicht) genannt, bei der jedes Attribut des strukturierten Typs auf eine Spalte der Tabelle abgebildet wird. Zusätzlich hat jede typisierte Tabelle eine implizit generierte Spalte (selbst-referenzierende Spalte genannt), die es erlaubt, einen eindeutigen

Identifizierer für jede Zeile zu speichern. Typisierte Tabellen können in *Subtable-Supertable*-Beziehungen stehen, die die Typen-hierarchiebeziehungen ihrer assoziierten Typen widerspiegeln.

SQL:1999 stellt auch einen speziellen Typkonstruktor REF bereit, der zu einem strukturierten Typ *T* einen Referenztyp REF(*T*) generiert. Ein gegebener Referenztyp ist stets mit einem strukturierten Typ assoziiert. Immer, wenn eine Zeile in eine typisierte Tabelle des Typs *T* eingefügt wird, wird ein Wert vom Typ REF(*T*) (entweder vom System generiert oder vom Benutzer geliefert) in die selbstreferenzierende Spalte dieser Zeile eingetragen. Der Wert eines Referenztyps identifiziert entweder eindeutig eine Zeile in einer typisierten Tabelle, oder aber er identifiziert nichts, was heißen könnte, dass es sich um eine ungebundene Referenz handelt, die übrig blieb, als die Zeile, die sie identifizierte, gelöscht wurde.

Das Typ-System von SQL:1999 ist orthogonal in dem Sinne, dass jede Art von Datentyp, also vordefinierte, konstruierte oder benutzerdefinierte, überall dort benutzt werden kann, wo ein Datentyp erlaubt ist. Zum Beispiel kann das Attribut eines UDT ein ARRAY-Typ sein, und der Elementtyp eines ARRAY ein UDT.

SQL:1999 stellt zusätzlich zu den von SQL:1992 übernommenen Prädikaten drei neue bereit: Das SIMILAR-Prädikat, das DISTINCT-Prädikat und das Typ-Prädikat. Das SIMILAR-Prädikat erlaubt anspruchsvolle *String-Matching*-Operationen, die auf regulären Ausdrücken beruhen, und die mächtiger sind als die einfachen Vergleiche, die das LIKE-Prädikat ermöglicht, das schon in den vorhergehenden Versionen unterstützt wurde. Das DISTINCT-Prädikat ähnelt in seinen Eigenschaften dem gewohnten UNIQUE-Prädikat; der wesentliche Unterschied ist, dass zwei NULL-Werte als ungleich betrachtet werden, wenn man das UNIQUE-Prädikat benutzt, und als gleich, wenn man DISTINCT verwendet (obwohl sie weder gleich noch ungleich sind). Damit erfüllen zwei NULL-Werte das DISTINCT-Prädikat nicht. Das Typ-Prädikat schließlich erlaubt es zu testen, ob der Laufzeit-Typ eines Ausdrucks in einer Liste spezifizierter Typen enthalten ist.

Mit SQL:1999 ist es erstmals möglich, rekursive Anfragen zu formulieren. Das ist nützlich für Anwendungen wie zum Beispiel Stücklistenverarbeitung.

SQL:1999 kennt auch das Konzept der Sicherungspunkte (*Savepoints*), das bereits in vielen Produkten implementiert ist. Ein Sicherungspunkt entspricht insofern einer Subtransaktion, als eine Applikation Aktio-

nen zurücksetzen kann, die nach einem Sicherungspunkt begonnen wurden, ohne dass die Aktionen der gesamten Transaktion zurückgesetzt werden müssen.

SQL:1999 erweitert das Privilegienmodell von SQL:1992 um den Begriff Rollen (*roles*). Privilegien können an Rollen vergeben werden wie an jeden normalen Benutzer, und Rollen können an normale Benutzer und an andere Rollen vergeben werden. Diese geschachtelte Struktur kann die oft schwierige Aufgabe der Sicherheitskontrolle in einer Datenbankumgebung dadurch enorm vereinfachen, dass man den Zugriff auf bestimmte Objektklassen nicht auf der Basis individueller Zugriffsrechte regelt, sondern auf der Basis von Job-Verantwortlichkeiten. So braucht zum Beispiel die Gruppe der Datenbankadministratoren andere Privilegien als die Benutzer der Lohn-Anwendungen.

SQL:1999 stellt eine wichtige Datenbankfunktionalität in Form von Triggern bereit. Mit Triggern kann ein Datenbankdesigner das Datenbanksystem anweisen, automatisch bestimmte Operationen jedes Mal dann durchzuführen, wenn eine Applikation ein INSERT, UPDATE oder DELETE auf einer bestimmten Tabelle durchführt. Die spezifizierte Operation kann einmal für die INSERT-, UPDATE- oder DELETE-Anweisung ausgeführt werden, oder individuell für jede betroffene Zeile.

Seit der Veröffentlichung von SQL:1999 wurden drei weitere inkrementelle Teile veröffentlicht, die als Teile von SQL:1999 betrachtet werden. Es sind das Teil 9: SQL/MED (Management of External Data), Teil 10: SQL/OLB (*Object Language Bindings*) und Teil 13: SQL/JRT (*SQL Routines and Types Using the JAVA™ Programming Language*). SQL/MED wird detailliert in Kapitel 4 beschrieben, SQL/OLB und SQL/JRT in Kapitel 5. Ein Addendum zu SQL:1999 standardisiert Erweiterungen zu SQL, die es erlauben, komplexe statistische und Datenanalyse-Funktionen als Teile normaler SQL-Anweisungen zu spezifizieren und sie auf der SQL-Maschine selbst zu berechnen. Beispiele solcher Funktionalität sind die Berechnung von Rangfolgen, kumulativen und inversen Verteilungen, Standardabweichungen, Varianzen, Co-Varianzen und linearen Regressionen.

Eine Übereinstimmung (*conformance*) mit der Norm SQL:1999 erfordert sowohl die Implementierung einer Teilmenge des Funktionsumfangs von Teil 2: SQL/Foundation (informell „Kern-SQL“ genannt) als auch die Bindung an eine der Host-Sprachen, wie sie in Teil 5: SQL/Bindings spezifiziert sind. Die Übereinstimmung mit anderen Teilen ist optional. Obwohl zurzeit kein Produkt eine Übereinstimmung mit SQL:1999 in Anspruch

nimmt, wissen die Autoren von einigen Anstrengungen, dieses Ziel zu erreichen.

### 3 Neue Funktionalität in SQL:2003

Die Revision des Jahres 2003 enthält den gesamten Funktionsumfang von SQL:1999, aber auch einen neuen Teil, Teil 14<sup>1)</sup>: SQL/XML (*XML-Related Specifications*). Dieser Teil wird etwas genauer in Kapitel 6 beschrieben. Zusätzlich gibt es eine kleinere Reorganisation der Teile, die von SQL:1999 geerbt wurden. Ein wesentlicher Teil von Teil 2: SQL/Foundation, in dem es um „*Information Schema*“ und „*Definition Schema*“ geht, bildet nun einen eigenen Teil, Teil 11: SQL/Schemata. Teil 5: SQL/Bindings wurde überflüssig, weil das enthaltene Material in Teil 2: SQL/Foundation integriert wurde. Alle neun Teile der Revision der SQL-Norm wurden 2003 als SQL:2003-12 veröffentlicht.

SQL:1999 enthält in großem Umfang neues Material, dessen Spezifikation sich als schwierig erwiesen hat. Als Folge davon entdeckten Entwickler, die die neue Funktionalität realisieren wollten, Probleme in den Spezifikationen, die oft genug nicht eindeutig waren oder sogar nachweislich falsch. Aus diesem Grund wurde bei der nächsten Revision sehr viel Wert auf die Fehlerkorrektur gelegt und weniger auf die Erweiterung des Funktionsumfangs.

Trotzdem ist auch neue Funktionalität bei SQL:2003 hinzugekommen, vor allem dadurch, dass Sprachelemente in die Norm aufgenommen wurden, die es in einigen Produkten bereits gab, und die von den Kunden gut aufgenommen worden waren.

Neue Sprachelemente in SQL:2003 sind:

- die MERGE-Anweisung
- Identitätsspalten
- generierte Spalten
- Generatoren für Folgen
- OLAP (*on-line analytic processing*)-Erweiterungen (diese wurden erstmals im Addendum zu SQL:1999 spezifiziert)
- CREATE TABLE ... AS Query
- TABLESAMPLE
- Mehrfach-Zuweisungen
- MULTISSET-Typen
- der Typ BIGINT (*big integer*)
- verbesserte *Savepoint*-Behandlung
- erweiterte Diagnostik.

Es folgen kurze Erläuterungen der Hauptmerkmale einiger Erweiterungen. In allen

<sup>1)</sup> Aus historischen Gründen sind die Nummern nicht fortlaufend, und einige Nummern sind unbenutzt. In SQL:2003 fehlen zum Beispiel die Teile 5, 6, 7, 8 und 12.

Fällen ist die komplette Spezifikation komplizierter, als es unsere Beispiele zeigen.

### MERGE-Anweisung

SQL hat drei wohlbekannte Operatoren zur Aktualisierung der Datenbank, INSERT, UPDATE und DELETE. In jedem Fall ist das einzige Zielobjekt der Operation eine Tabelle, und in jedem Fall betrifft der Effekt eine Anzahl Zeilen: INSERT fügt Zeilen ein, UPDATE ersetzt einige Spaltenwerte in vorhandenen Zeilen und DELETE löscht einige der vorhandenen Zeilen.

Manchmal finden sich in einer Applikation einheitlich formatierte Datensätze (also tabellarische Daten) in der Form, dass einige Zeilen in der Ziel-Tabelle ein Gegenstück haben, andere hingegen nicht. Die Zeilen ohne Gegenstück sollen eingefügt werden, die Zeilen mit Gegenstück sollen ihr Gegenstück aktualisieren.

Was ist nun ein passendes Gegenstück? Betrachten wir ein einfaches Beispiel und ignorieren, dass es etwas künstlich anmutet. Die Tabelle PUNKTE enthält die Punkte, die Studenten bei verschiedenen Aufgaben in verschiedenen Kursen eines Jahres erreicht haben. Wir zeigen im folgenden Beispiel nur ein paar Zeilen.

KURS	AUFGABEN_NR	STUDENTEN_NR	PUNKTE
C1	1	S1	56
C1	2	S1	63
C1	1	S2	88
C1	2	S2	91
C2	1	S4	49

**Tabelle 1:** Tabelle PUNKTE

Die Spalten KURS, AUFGABEN\_NR und STUDENTEN\_NR bilden den Primärschlüssel der Tabelle. Also dürfen keine zwei Zeilen der Tabelle in diesen Spalten die gleiche Wertekombination haben. Eine gegebene Kombination von Kurs, Aufgabennummer und Studentenummer ist also hinreichend (und notwendig), um eine einzelne Zeile in der Tabelle zu identifizieren, wenn denn eine solche Zeile existiert.

Angenommen, ein zentrales Büro der Universität sammelt jede Woche Informationen von allen Abteilungen in der gezeigten Form. Einige der Zeilen sind dabei Korrekturen bereits vorhandener Sätze, andere sind neue benotete Aufgaben. Hier ein Beispiel:

KURS	AUFGABEN_NR	STUDENTEN_NR	PUNKTE
C1	1	S1	57
C1	3	S1	69
C1	2	S2	81
C2	1	S3	95

**Tabelle 2:** Tabelle NEUE\_PUNKTE

Die erste Zeile in NEUE\_PUNKTE hat ein passendes Gegenstück in PUNKTE und die dritte gezeigte Zeile ebenfalls: die Kombination der Werte im Primärschlüssel passt nämlich zu einer existierenden Zeile in PUNKTE. Die zweite und vierte Zeile haben dagegen kein passendes Gegenstück.

Man beachte, dass man keine Vereinigung (mittels UNION) von PUNKTE und NEUE\_PUNKTE bilden und das Ergebnis dann an PUNKTE zuweisen kann. Das würde nämlich zwei kollidierende Zeilen für den Studenten S1 bei der Aufgabe 1 im Kurs C1 zur Folge haben. Hier kann nun die neue MERGE-Anweisung benutzt werden, um gleichzeitig die falschen Noten zu korrigieren und die neuen hinzuzufügen:

```

MERGE INTO PUNKTE AS P
USING NEUE_PUNKTE AS N
ON P.KURS = N.KURS
AND P.AUFGABEN_NR = N.AUFGABEN_NR
AND P.STUDENTEN_NR = N.STUDENTEN_NR
WHEN MATCHED THEN UPDATE
SET PUNKTE = N.PUNKTE
WHEN NOT MATCHED THEN INSERT
VALUES (N.KURS, N.AUFGABEN_NR,
N.STUDENTEN_NR, N.PUNKTE)

```

Beachtenswert ist, dass NEUE\_PUNKTE durch einen beliebigen SQL-Query-Ausdruck ersetzt werden kann, dass also jede Tabelle, die aus der Datenbank abgeleitet werden kann, als Eingabe für MERGE dienen kann. Bemerkenswert ist ebenfalls, dass die USING-Tabelle nicht das gleiche Format haben muss wie die INTO-Tabelle. Sie muss nur eine oder mehrere Spalten für die Trefferermittlung enthalten sowie die Spalten, die für INSERT und UPDATE erforderlich sind. Der in SQL bereits versierte Leser ist eingeladen, gedanklich die Möglichkeiten durchzugehen, die dieses neue Sprachelement bietet.

### Identitätsspalten

Betrachten Sie die STUDENTEN\_NR der PUNKTE-Tabelle aus dem MERGE-Beispiel. Wie wurde entschieden, dass ein bestimmter Student durch die Registrierungsnummer „S1“ identifiziert werden soll, ein anderer durch „S2“ und so weiter. Natürlich hat die Universität einen Generierungsprozess, der die Nummern so erzeugt, dass keine zwei Studenten

die gleiche Kennung bekommen. Man kann SQL dazu veranlassen, diese Zuordnung vorzunehmen, indem man das neue Merkmal „Identitätsspalte“ benutzt.

Bei der Spalte STUDENTEN\_NR in PUNKTE handelt es sich um einen externen Schlüssel, der auf eine Tabelle STUDENT zeigt, in der wiederum in jeder Zeile Information steht, die die Universität über die Studenten braucht. STUDENT hat ebenfalls eine Spalte STUDENTEN\_NR, die der Primärschlüssel von STUDENT ist, und die sicherstellt, dass keine zwei Studenten die gleiche STUDENTEN\_NR haben. Wenn bei der Erzeugung der Tabelle STUDENT die Spalte STUDENTEN\_NR als Identitätsspalte deklariert wird, dann erzeugt das System bei INSERT-Operationen für diese Spalte automatisch einen Wert und stellt sicher, dass kein bereits existierender Wert jemals wiederverwendet wird.

Bei der Verwendung dieses Merkmals müssen allerdings die folgenden Einschränkungen akzeptiert werden:

- Die STUDENTEN\_NR muss eine Nummer sein; eine Ziffernfolge mit vorangehenden Buchstaben (wie im vorangehenden Beispiel) ist nicht zulässig. Da aber solche Konstruktionen wohl nur in Lehrbüchern vorkommen, nicht aber im wahren Leben, sollte das kein Problem sein.
- Die willkürliche Auswahl der Zahlen durch das System muss toleriert werden. Bestimmte Nummernkreise für bestimmte Studentengruppen können zum Beispiel nicht vorgegeben werden.

### Generierte Spalten

Dieses neue Merkmal erlaubt es, in einer bestimmten Spalte einer Basistabelle einen Wert als Rechenergebnis einer vorgegebenen Formel einzufügen und nicht auf dem herkömmlichen Weg, d. h. über Werte, die beim Einfügen einer Zeile in die Tabelle explizit spezifiziert werden. Üblicherweise werden bei diesem Vorgehen Spalten in der einzufügenden Zeile referenziert. Beispielsweise könnte eine Spalte GESAMTZAHLUNG eine generierte Spalte sein, die über die Formel LOHN + BONUS gefüllt wird.

Warum sollte nun jemand Speicherplatz für eine redundante Information verwenden, die man durch eine Berechnung im Bedarfsfall ganz einfach selbst gewinnen kann? Eine mögliche Antwort ist, dass manchmal Zeit wichtiger ist als Speicherplatz. Normalerweise ist die Query-Auswertung auf der Basis gespeicherter (statt abgeleiteter) Daten schneller, vor allem, wenn ein Index für eine solche Spalte vorhanden ist. Soll beispielsweise die GESAMTZAHLUNG häufig in absteigender Reihenfolge angezeigt werden, erhält

man das Ergebnis wesentlich schneller, wenn ein Index für diese Spalte vorliegt.

### Generatoren für Folgen

Gehen wir zurück zu der PUNKTE-Tabelle aus unserem MERGE-Beispiel und schauen uns die Spalte AUFGABEN\_NR an. Sie wirft ein ähnliches Problem auf wie die STUDENTEN\_NR bei unseren Ausführungen über Identitätsspalten, allerdings mit dem Unterschied, dass Aufgabennummern nur innerhalb des KURSES eindeutig sind. Wenn es also eine Tabelle AUFGABEN gäbe, dann müsste der Primärschlüssel eine Kombination aus KURS und AUFGABEN\_NR sein. Das Merkmal Identitätsspalte kann deshalb nicht in der gleichen Weise für eine automatische Nummerngenerierung genutzt werden, wie wir das bei der Studentenummer gemacht haben.

Hier helfen nun Folgeneratoren, eine neue Klasse von Datenbankobjekten. Ein Folgenerator wird erzeugt durch die Ausführung des Befehls CREATE SEQUENCE GENERATOR. Ein Folgenerator hat einen Startwert – einen INTEGER-Wert – und einen Steigerungswert. Sei *SG* ein solcher Folgenerator mit der Steigerung *i*; dann addiert der Befehl NEXT VALUE FOR *SG* den Wert *i* zu dem bisherigen Wert von *SG* und gibt das Resultat zurück.

Um nun unsere Kursnummern automatisch zu generieren, erzeugen wir einen Folgenerator für jeden Kurs. Immer dann, wenn Studenten in einem Kurs eine neue Aufgabe erhalten, erzeugt ein Aufruf von NEXT VALUE FOR für den Folgenerator dieses Kurses die erforderliche neue Aufgabennummer.

### CREATE TABLE ... AS Query

In verschiedenen Produkten gab es dieses Merkmal bereits seit einiger Zeit. Jetzt ist es auch Bestandteil der Norm. Betrachten wir irgendeine Anfrage *Q*, die eine Tabelle liefert, in der keine zwei Spalten den gleichen Namen haben; dann hat CREATE TABLE *TN* AS *Q* den Effekt, eine Basistabelle *TN* zu erzeugen und dieser Tabelle den Wert von *Q* zuzuweisen. Wenn es später notwendig werden sollte, den Inhalt der Tabelle *TN* zu verändern, dann ist das eine Angelegenheit geeigneter DELETE-, UPDATE- und INSERT-Anweisungen. Will man beispielsweise erreichen, dass *TN* den aktuellen Zustand der Datenbank widerspiegelt, wie er sich bei einer Neuauswertung der Query *Q* ergäbe, dann kann man das erreichen, indem man mit DELETE FROM *TN* die vorhandenen Tupel in *TN* löscht und mit INSERT INTO *TN* *Q* die Anfrage *Q* neu auswertet und das Ergebnis in *TN* einfügt.

### TABLESAMPLE

Statistiker benutzen oft Stichprobenverfahren, um dadurch den Aufwand zu vermeiden, eine vollständige Grundgesamtheit (zum Beispiel eine umfangreiche Tabelle) zu analysieren. Die neue TABLESAMPLE-Klausel, die als Tabellenreferenz in der FROM-Klausel auftritt, erlaubt eine zufällige Auswahl von Zeilen, die sonst nur durch Sub-Ausdrücke einer nicht ganz trivialen Query zu gewinnen wären. Die statistische Analyse einer eingeschränkten Ergebnismenge kann wesentlich schneller beendet werden und trotzdem alle Informationen liefern, die für Aufgaben wie Entscheidungsfindung benötigt werden.

### Mehrfach-Zuweisung

Zuweisungen werden in SQL zum einen dazu verwendet, in UPDATE- und MERGE-Anweisungen jeder Zeile, die von der Anweisung betroffen ist, die entsprechenden Spaltenwerte zuzuweisen, zum anderen, um lokale Variablen in Anwendungsprogrammen, Stored procedures und benutzerdefinierten Funktionen mit Werten zu besetzen. In SQL:1999 ist diese Zuweisung nur von „einfacher“ Art, das heißt, das Ergebnis der Auswertung eines einzelnen skalaren Ausdrucks kann nur einem einzelnen Ziel zugewiesen werden.

Mehrfach-Zuweisungen erlauben es nun, jedes einzelne Feld eines *n*-Tupels mit skalaren Werten dem entsprechenden Element in einer Liste von Zielvariablen zuzuweisen. Die skalaren Ausdrücke, die das *n*-Tupel bilden, werden dabei alle ausgewertet, bevor irgendeine Zuweisung stattfindet. SQL-Benutzer kennen dieses Konzept bereits von FETCH- und SELECT INTO-Anweisungen; nun ist es auch in regulären Zuweisungen verfügbar.

### MULTISET-Typen

Eine Multimenge (*multiset*) ist eine Kollektion (in Mathematikerkreisen gewöhnlich „*bag*“ genannt). Eine Multimenge unterscheidet sich von einer Menge (*set*) dadurch, dass das gleiche Element mehr als einmal auftreten kann. So könnten zum Beispiel die Primfaktoren von 24 als Multimenge ausgedrückt werden: {2, 2, 2, 3}. Wie bei Mengen haben Elemente konzeptionell keine Ordnungsposition. Im Prinzip ist auch eine SQL-Tabelle eine Multimenge (von Zeilen desselben Zeilentyps). Operationen auf Multisets entsprechen den bekannten Mengen-Operatoren Vereinigung, Differenz und Durchschnitt. SQL unterstützt einige von ihnen mit seinen Operatoren UNION ALL, EXCEPT ALL und INTERSECT ALL.

Der neue Typ-Konstruktor MULTISET in SQL:2003 erlaubt es, Multiset-Typen in jeder

Deklaration dort zu benutzen, wo ein Datentyp angegeben werden muss, wie zum Beispiel bei einer Spaltendefinition, einer Attributdefinition, einer Parameterdefinition oder dem Ergebnistyp einer benutzerdefinierten Funktion.

Der möglicherweise interessanteste Aspekt des MULTISET-Typs besteht darin, dass eine Tabellenspalte vom Typ MULTISET sein kann, wobei der Element-Typ ein Zeilentyp ist; dadurch unterstützen wir geschachtelte Tabellen (*nested tables*), ein interessantes Konzept, das intensiv in den späten 70er- und in den 80er-Jahren untersucht wurde.

## 4 SQL/MED

Obwohl die SQL-Norm in vielen marktgängigen relationalen Datenbanksystemen implementiert ist, und diese Systeme enorm umfangreiche Datenbestände verwalten, sollte es den Leser nicht verwundern, dass gewaltige Datenmengen immer noch in gewöhnlichen Betriebssystem-Dateien, in Netzwerk- und hierarchischen Datenbanken oder in historisch gewachsenen Speichersystemen verwaltet werden. Die Normungsgremien haben das erkannt und entsprechende SQL-Erweiterungen erarbeitet, die in Teil 9: SQL/MED (*Management of External Data*) zusammengefasst sind. Dieser Teil widmet sich Konstrukten, die es dem Benutzer erlauben, neben den SQL-Daten so genannte externe Daten abzufragen und zu manipulieren, d. h. Daten, die zunächst einmal nicht unter der Kontrolle des benutzten SQL-Datenbanksystems stehen.

SQL/MED kennt zwei unterschiedliche Aspekte des Zugriffs auf externe Daten. Beim ersten Aspekt geht es um die Fähigkeit, die SQL-Schnittstelle zu benutzen, um auf Nicht-SQL-Daten (*foreign data*) zuzugreifen, und diese, wenn gewünscht, mit SQL-Daten zu verbinden. Eine Applikation übergibt zu diesem Zweck lediglich eine einzige SQL-Anfrage an den SQL-Server, der seinerseits Daten aus verschiedenen Quellen referenziert. Da es für die Applikation unerheblich ist, dass einige der Daten weder lokal gespeichert noch lokal verwaltet werden, ist es die Aufgabe des SQL-Servers, die Anfrage in geeignete Teile (in der Norm *requests* genannt) zu zerlegen, die dann an die einzelnen Datenquellen weitergegeben werden. In der Norm ist nicht vorgegeben, wie diese Zerlegung aussehen muss. In ihr ist nur das Zusammenspiel zwischen dem SQL-Server und einer Hüll-Schicht für Fremd-Daten (*foreign-data wrapper*) spezifiziert, das der Zerlegung der Anfrage und deren nachfolgender Ausführung zugrunde liegt.

Beim zweiten Aspekt des Zugriffs auf externe Daten geht es um ein spezielles Datenver-

waltungsproblem: Große Mengen unternehmenskritischer Daten liegen häufig in Dateisystemen, wobei es sich oftmals um (wenigstens aus SQL-Sicht) nicht-traditionelle Daten handelt, wie zum Beispiel Konstruktionspläne, Fotografien oder andere Medien. In den meisten Fällen können existierende Applikationen nur dann unverändert weiterbestehen, wenn diese Daten in ihrem Dateisystem verbleiben; es wäre jedoch oft von Vorteil, Informationen über diese Dateien in einer Datenbank zu speichern, weil eine Datenbank einfacher befragt werden kann. Bei diesem typischen Szenario ist es jedoch wünschenswert, zusammengehörige Daten in der Datenbank und im Dateisystem nur synchron zu verändern. Die Problematik dabei besteht vor allem darin, dass Datenbank und Dateisystem über unterschiedliche Schnittstellen anzusprechen sind und dass insbesondere unterschiedliche Autorisierungsverfahren berücksichtigt werden müssen. SQL/MED adressiert diese Problematik durch die Einführung eines neuen Datentyps namens DATALINK.

Leser, die sich für eine detailliertere Beschreibung der Funktionalität von externen Daten und von Datalinks interessieren, als sie im Folgenden gegeben wird, seien auf [3, 4] verwiesen.

## Externe Daten

Wenn externe Daten nahtlos in den Kontext von SQL passen sollen, müssen sie als relationale Tabellen dargestellt werden. SQL/MED gestattet es, Daten, die außerhalb des SQL-Servers gespeichert werden, dem SQL-Server gegenüber als externe Tabellen (*foreign tables*) darzustellen. So kann zum Beispiel eine im Dateisystem gespeicherte Preisliste für ein Automobil als (externe) Tabelle dargestellt werden, etwa mit Spalten für den Hersteller, das Modell, das Jahr, den Preis usw. Externe Datenquellen stellen oft mehr als nur eine Kollektion derartiger Datensätze bereit, wobei auf alle diese über eine einzige Netzwerk-Verbindung zugegriffen werden kann. Daher wird in SQL/MED das Konzept des externen Servers (*foreign server*) eingeführt, das den Zugriff auf eine Menge externer Tabellen ermöglicht. Zum Beispiel könnte eine bestimmte Web-Seite unterschiedliche Preislisten für LKWs, Autos, Motorräder usw. bereitstellen, die jeweils als eigene (externe) Tabelle nach außen dargestellt werden sollen. Wenn nun der SQL-Server eine Anfrage in Anfrage-Fragmente zerlegt, wird jedes Fragment an den externen Server gesendet, der für die externe Tabelle zuständig ist, die in dem betreffenden Fragment referenziert wird. In der ersten Version von SQL/MED konnte ein Fragment nicht mehr als eine Tabelle referenzieren. Nach

SQL/MED:2003 hingegen ist es erlaubt, einem externen Server Anfrage-Fragmente zu übergeben, die mehrere externe Tabellen referenzieren und die darüber hinaus Prädikate und komplexe Ausdrücke enthalten.

Wenn sich verschiedene Datenquellen eine gemeinsame Schnittstelle teilen, dann sollte nur ein einzelnes Code-Modul erforderlich sein, um auf alle diese Quellen (von denen jede durch einen externen Server repräsentiert wird) zuzugreifen. Dieses gemeinsame Code-Modul manifestiert sich in SQL/MED als die Fremd-Daten-Hülle (*foreign-data wrapper*). So kann zum Beispiel eine einzige Fremd-Daten-Hülle benutzt werden, um auf mehrere Web-Seiten zuzugreifen, von denen wiederum jede als externer Server betrachtet wird. In SQL/MED werden keine Aussagen über Voraussetzungen für die externen Server getroffen. Daher könnte jede beliebige Datenquelle die Rolle eines externen Servers einnehmen, solange es nur eine geeignete Fremd-Daten-Hülle gibt, die die Datenquelle verkapselt und die in der Lage ist, die gespeicherten Daten in Tabellenform zu präsentieren. Die Norm SQL/MED spezifiziert neue DDL-Anweisungen, mit denen externe Tabellen, externe Server und Fremd-Daten-Hüllen erzeugt werden können. Sie standardisiert auch eine Schnittstelle (im Wesentlichen eine Ansammlung von Funktionen), über die ein SQL-Server und eine Fremd-Daten-Hülle ihre Aufgaben erfüllen.

Im Augenblick kann auf externe Tabellen nur lesend (*read-only*) zugegriffen werden; Daten, die durch diese Tabellen repräsentiert werden, können nicht unmittelbar über den SQL-Server durch UPDATE, INSERT oder DELETE verändert werden. Solche Operationen müssen vielmehr an dem externen Server selbst ausgeführt werden.

## Datalinks

Der neue DATALINK-Typ erlaubt es, Dateireferenzen dauerhaft in einer Datenbank zu speichern. Solche Dateireferenzen (in den SQL-Norm werden sie *Datalinks* genannt) haben die Form einer URL. Es sei noch einmal betont, dass in der Datenbank nur Referenzen auf Dateien gespeichert werden, wohingegen die Dateien selbst in dem Dateisystem gespeichert bleiben, in dem sie ursprünglich abgelegt wurden.

Ein Datalink-Wert wird in einer Tabellen-Spalte des Typs DATALINK nicht anders gespeichert als zum Beispiel ein ganzzahliger Wert in einer Spalte vom Typ INTEGER. Ein Datalink-Wert kann nicht nur als Wert in einer Spalte auftreten, sondern auch als Attribut-Wert eines UDT.

SQL/MED erlaubt eine Vielzahl von Optionen, die für Instanzen vom Typ DATALINK spezifiziert werden können. Mit diesen Optionen kann bestimmt werden, wie streng die Datenbank die zugehörige Datei kontrolliert. Die Möglichkeiten reichen von „keine Kontrolle“ (die Datei braucht nicht einmal zu existieren) bis zu „volle Kontrolle“ (wo das Löschen des Datalink-Wertes aus der Datenbank zum physikalischen Löschen der Datei führt). Um eine Kontrolle über die aktuelle Datei auszuüben, greift das Datenbank-System auf die Hilfe eines Datalinkers zurück. Der Datalinker befindet sich über dem Dateisystem; er fängt jeden Dateizugriff einer Applikation ab und untersucht, ob sie die erforderlichen Zugriffsrechte auf die Datei besitzt. Die Applikation selbst kann diese Zugriffsrechte vorweg von dem Datenbanksystem bekommen, das den Datalink speichert, der die fragliche Datei referenziert. Datalink-Werte können aus der Datenbank ausgelesen, aber auch eingefügt, aktualisiert und gelöscht werden, und zwar mit der gleichen Transaktionssemantik wie bei allen anderen Daten im Datenbanksystem.

## 5 SQL/OLB und SQL/JRT

SQL/OLB (*Object Language Bindings*) und SQL/JRT (*SQL Routines and Types Using the Java™ Programming Language*) stellen zwei von drei technischen Regelwerken dar, die zusammen eine enge Verbindung zwischen Java und SQL herstellen. Das dritte, JDBC™, ein API für Java-Applikationen zum Ausführen von dynamischem SQL, ist vergleichbar mit SQL/CLI. Im Folgenden beschreiben wir SQL/OLB und SQL/JRT, ihre Beziehung zu JDBC sowie die Erweiterungen, die in ihre letzten Revisionen eingebracht wurden.

### SQL/OLB

Als SQL/OLB entwickelt wurde, war JDBC eine De-facto-Norm, und es besteht eine sehr enge Beziehung zwischen SQL/OLB und JDBC. SQL/OLB, manchmal auch SQLJ Teil 0 genannt, macht eingebettetes SQL für Java-Applikationen verfügbar. SQL/OLB bietet unter anderem folgende Vorteile:

- Schnellere Applikationen-Entwicklung. Es wird ein höheres Abstraktionsniveau als bei JDBC bereitgestellt, und zwar auf der Basis von statischem SQL. SQL/OLB bietet eine Typ-Überprüfung zur Vorübersetzungszeit (von SQL/OLB Übersetzungszeit genannt) und eine Benachrichtigung über ungültige SQL-Syntax.
- Schnellere Ausführung wegen der Benutzung von statischem (anstelle von dynamischem) SQL.

Zu den Anforderungen an SQL/OLB gehörte es von Anfang an, dass eine Java-Applikation in der Lage sein sollte, dynamisches und eingebettetes SQL so zu mischen, wie es gebraucht wird; dabei sollten in den unterschiedlichen SQL-Stilarten geschriebene Code-Stücke sowohl Verbindungen zur SQL-Implementierung als auch zu Cursors (d. h. die JDBC-definierte `java.sql.connection` bzw. das `java.sql.ResultSet`) gemeinsam benutzen. Ein starkes Motiv für eine derartige Interoperabilitäts-Anforderung war eine SQL/OLB-Referenz-Implementierung, die öffentlich verfügbar gemacht wurde und in der die Unterstützung für eingebettetes SQL in einer JDBC-basierten Laufzeitumgebung realisiert worden war.

Betrachten wir als Beispiel von SQL/OLB die folgende eingebettete SQL-Anweisung:

```
#sql [connCtxt] {INSERT INTO
  DEPARTMENT(DName, DMgr, DId)
  VALUES(:dptName, :dptMgr, :dptID)};
```

Diese Anweisung wird mit `#sql` am Anfang und mit einem Semikolon am Ende gegen den umgebenden Java-Code abgegrenzt. `connCtxt` gehört zu einer von SQL/OLB definierten Klasse; sie steht in Beziehung zu `java.sql.Connection` von JDBC, die ihrerseits eine Verbindung zu einer SQL-Implementierung darstellt. `dptName`, `dptMgr` und `dptID` enthalten den Namen, den Managernamen und den Identifizierer für die neue Abteilung.

Wenn man ausschließlich JDBC verwendet, benötigt man unter Umständen zahlreiche JDBC-Methoden-Aufrufe, um gerade einmal das Äquivalent für eine einzige mit `#sql` eingebettete Anweisung zu bekommen. Die einfache INSERT-Anweisung weiter oben wäre höchstwahrscheinlich als Aufruf von fünf JDBC-Methoden zu kodieren; der Leser kann sich leicht vorstellen, dass ein INSERT in eine Tabelle mit 100 Spalten über 100 einzelne JDBC-Methodenaufrufe benötigen würde anstatt einer einzigen eingebetteten `#sql`-Anweisung.

## SQL/JRT

Die andere Norm, die Java und SQL verbindet, ist SQL/JRT. SQL/JRT umfasst zwei eng verwandte Fähigkeiten. Die erste betrifft den Aufruf statischer Java-Methoden als SQL-aufzurufbare Routinen. SQL/JRT erlaubt CALL-Anweisungen, um aus SQL Java-Void-Methoden aufzurufen, oder auch benutzerdefinierte SQL-Funktionen, die in Java geschrieben sind. Die zweite Fähigkeit besteht darin, dass SQL/JRT die SQL-CREATE TYPE-Anweisung um die Möglichkeit erweitert, Abbildungen zwischen Java-Klassen und darauf basierenden SQL UDTs zu deklarieren. Die Felder der

Java-Klasse werden die Attribute des SQL UDT, und die Klassenmethoden werden aufrufbar als benutzerdefinierte SQL-Methoden.

## Letzte Erweiterungen in SQL/OLB und SQL/JRT

SQL/OLB:2003 und SQL/JRT:2003 haben ihre für die Norm verbindlichen Verweise (so genannte normative Verweise) auf JDBC 3.0 [5] und auf die zweite Fassung der Java-Sprachspezifikation [6] aktualisiert.

Durch die Benutzung von JDBC 3.0 als normativen Verweis ergibt sich eine größere gemeinsame Schnittmenge zwischen der De-facto-Norm JDBC und der De-jure-Norm SQL. Zusätzlich zu den aktuelleren normativen Verweisen umfassen die Erweiterungen von SQL/OLB:

- Unterstützung des SQL-Datentyps ARRAY und des SQL/MED-Datentyps DATALINK als Java-Klassen `java.sql.Array` bzw. `java.net.URL`.
- Unterstützung von SQL-Sicherungspunkten mit der Möglichkeit, sie in einer eingebetteten SQL-Anweisung zu setzen, sie freizugeben und ein Rücksetzen auf einen Sicherungspunkt durchzuführen.
- Angenommen, eine SQL-aufzurufbare Routine gibt mehr als eine dynamische Ergebnismenge zurück. Dann nutzt SQL/OLB jetzt die Fähigkeit von JDBC 3.0, mehrere dieser Ergebnismengen gleichzeitig für navigierenden Zugriff offen zu halten.

Die jüngsten Erweiterungen von SQL/JRT umfassen:

- Unterstützung von SQL-aufzurufbaren Funktionen, deren Ergebnistyp ein SQL ARRAY oder MULTISSET ist.
- Unterstützung des DATALINK-Datentyps als Parametertyp oder als Attribut eines UDT.

Schließlich können Instanzen eines SQL UDT, der als Java-Klasse implementiert ist, an eine Java-Applikation zurückgegeben werden, indem man zum Beispiel eine SELECT-Anweisung entweder mit JDBC oder SQL/OLB benutzt; dabei bleiben die Klasseneigenschaften (*classiness*) dieser Daten erhalten. Wenn eine Instanzmethode aufgerufen wird, ob nun als SQL-aufzurufbare Methode oder direkt über einen Java-Aufruf, kann diese Java-Methode ihrerseits JDBC oder SQL/OLB benutzen, um dynamisches bzw. statisches SQL auszuführen. Wie gesagt, es besteht jetzt eine recht enge Verbindung zwischen Java und SQL.

## 6 SQL/XML

In der zweiten Hälfte des Jahres 2000 begann die Arbeit an einem neuen Teil von SQL, der SQL/XML genannt wurde. Diese Arbeit wurde angestoßen durch die steigende Verwendung von XML (*Extensible Markup Language*) beim Datenaustausch sowohl innerhalb von Organisationen als auch zwischen Organisationen.

Die erste Version von SQL/XML, die zusammen mit den anderen Teilen von SQL:2003 im Jahr 2003 eingeführt wurde, hat drei Hauptmerkmale: einen XML-Datentyp, Funktionen, um XML aus SQL-Daten zu erzeugen, und eine Abbildung von SQL-Tabellen auf XML-Dokumente. Diese Funktionalität erlaubt es Applikationen, XML-Dokumente in SQL-Datenbanken zu speichern oder XML-Daten aus traditionellen SQL-Daten zu erzeugen, die dann von XML-orientierten Applikationen benutzt werden können.

Leser, die an einer detaillierteren Beschreibung der Funktionalität von SQL/XML interessiert sind, als sie hier möglich ist, werden auf [7; 8] verwiesen.

### Der XML-Datentyp

SQL/XML erweitert das vorhandene Typ-System von SQL um den Datentyp XML. Instanzen dieses Datentyps können XML-Dokumente enthalten, individuelle Elemente und Folgen von Elementen (die keinen Wurzelknoten haben). Dieser Datentyp kann benutzt werden, um Spalten, Variable und Parameter zu definieren, und zwar in genau der gleichen Art und Weise wie bei jedem anderen SQL-Datentyp.

Zurzeit wird noch an einem Mechanismus gearbeitet, der dafür sorgt, dass eine Zeichenkette, die XML enthält, dazu benutzt werden kann, um einen XML-Wert in SQL zu erzeugen (eine *Parse*-Operation), und an einem Mechanismus, durch den ein XML-Wert in SQL benutzt werden kann, um (aus der Sicht von SQL) eine Zeichenkette zu erzeugen (eine *Serialisierungsoperation*). Ein XML-Dokument, das die *Parse*-Operation bestanden hat, kann ein zugehöriges XML-Schema haben, eine Dokumententypdefinition (DTD) oder es ist einfach nur wohlgeformt.

### Veröffentlichen von SQL-Daten im XML-Format

SQL/XML stellt mehrere Funktionen bereit, die auf SQL-Daten zugreifen und daraus Werte des neuen XML-Datentyps produzieren. Das folgende Beispiel benutzt `XMLELEMENT`, um ein XML-Element für jeden Angestellten eintrag zu erzeugen:

```
SELECT e.id,
       XMLELEMENT (NAME „Emp“,
                   XMLATTRIBUTES (e.id),
                   e.Iname
                   ) AS „result“
FROM employees e
WHERE ...;
⇒
```

ID	RESULT
1001	<Emp ID="1001">Smith</Emp>
1206	<Emp ID="1206">Martin</Emp>

XMLAGG ist eine Aggregierungsfunktion, die einen einzelnen XML-Wert aus einer Gruppe von XML-Werten erzeugt. In diesem Beispiel benutzen wir XMLAGG, um ein Abteilungs-element zu erzeugen, das für jeden Angestellten dieser Abteilung ein Element enthält.

```
SELECT XMLELEMENT
       (NAME „Department“,
        XMLATTRIBUTES
        (e.dept AS „name“),
        XMLAGG
        (XMLELEMENT
         (NAME „emp“, e.Iname)
         ORDER BY e.Iname
         )
       ) AS „dept_list“
FROM employees e GROUP BY dept;
⇒
```

dept_list
<Department name="Accounting"> <emp>Smith</emp> <emp>Yates</emp> </Department> <Department name="Shipping"> <emp>Martin</emp> <emp>Oppenheimer</emp> </Department>

Zusammen mit XMLCONCAT und XMLFOREST erlauben es diese Operationen dem Benutzer, komplexe XML-Werte aus seinen SQL-Daten zu erzeugen. Mit der XMLCONCAT-Operation lassen sich zwei oder mehr XML-Werte verketteten; mit der XMLFOREST-Operation kann eine Folge von XML-Elementen erzeugt werden.

### Abbildung von Tabellen auf XML-Dokumente

SQL/XML definiert eine Abbildung von Tabellen auf XML-Dokumente. Diese Abbildung akzeptiert als Quelle eine einzelne Tabelle, alle Tabellen in einem SQL-Schema oder alle Tabellen in einem Katalog. Die Abbildung pro-

duziert zwei XML-Dokumente, eines mit den Daten der spezifizierten Tabellen und ein zweites mit dem XML-Schema, das den Inhalt des ersteren Dokumentes beschreibt. Eine EMPLOYEE-Tabelle ließe sich in der folgenden Art und Weise auf ein XML-Dokument abbilden:

```
<EMPLOYEE>
  <row>
    <EMPNO>000010</EMPNO>
    <FIRSTNAME>CHRISTINE</FIRSTNAME>
    <LASTNAME>HAAS</LASTNAME>
    <BIRTHDATE>1933-08-24</BIRTHDATE>
    <SALARY>52750.00</SALARY>
  </row>
  <row>
    <EMPNO>000020</EMPNO>
    <FIRSTNAME>MICHAEL</FIRSTNAME>
    <LASTNAME>THOMPSON</LASTNAME>
    <BIRTHDATE>1948-02-02</BIRTHDATE>
    <SALARY>41250.00</SALARY>
  </row>
  ...
</EMPLOYEE>
```

Hier wurde ein Wurzelement mit dem Namen der Tabelle erzeugt. Das Wurzelement enthält ein <row>-Element für jede Zeile der Tabelle. Jedes <row>-Element wiederum enthält eine Folge von Spaltenelementen, jeweils mit dem Namen der Spalte. Jedes Spaltenelement schließlich enthält einen Datenwert. Eine Variante dieser Abbildung (die hier nicht gezeigt wird) würde kein übergreifendes Wurzelement erzeugen, und jedes <row>-Element wäre ein <employee>-Element.

Die Operationen, mit denen man solche Abbildungen erzielen kann, sind von System zu System verschieden. Im Gegensatz zu anderen SQL-Merkmalen gibt es zurzeit noch keine verbindliche SQL-Anweisung für diesen Zweck. Wenn Benutzer solche Abbildungen vornehmen wollen, müssen sie auf die eine oder andere Art festlegen, wie etwa NULL-Werte repräsentiert werden sollen oder wie XML-Namensräume in dem zu erzeugenden XML-Dokument behandelt werden sollen.

## 7 Ausblick

Es lässt sich mit einiger Sicherheit vorher-sagen, dass man auch in absehbarer Zukunft an den SQL-Normen weiterarbeiten wird. Die Datenbankhersteller müssen sich auch in Zukunft großen Herausforderungen an die Funktionalität und Leistungsfähigkeit ihrer Produkte stellen, was wiederum die Fortentwicklung der Normen vorantreiben wird. Wenn wir uns aber einmal umschaun, dann scheinen XML, das zugehörige Datenmodell und die Arbeiten an einer Anfragesprache in eine andere Richtung zu deuten und für einige Aufregung zu sorgen; auf der anderen

Seite zeichnet sich aber nichts am Horizont ab, das die gleichen Probleme löst wie SQL und das diese Probleme so gut löst wie SQL. Solange es keine großen Umwälzungen in der Technologie gibt, wird SQL, so glauben wir, eine „Intergalaktische Datensprache“ bleiben (wie es Michael Stonebraker, einer der Pioniere der Datenbank-Technologie, einmal formuliert hat).

Die SQL-Normen haben sich etabliert als ausgereifte Normen mit breiter Akzeptanz, aber auch als Normen, die immer noch in der Lage sind, neue Technologien aufzunehmen, wie es sich bei SQL/MED, SQL/OLB, SQL/JRT und SQL/XML gezeigt hat.

### Literaturhinweise

- [1] Jim Melton und Alan Simon: SQL:1999 – Understanding Relational Language Components Morgan Kaufmann Publishers, Mai 2001.
- [2] Jim Melton: Advanced SQL:1999 – Understanding Object-Relational and Other Advanced Features Morgan Kaufmann Publishers, September 2002.
- [3] Jim Melton u. a.: SQL and Management of External Data. In: ACM SIGMOD Record, Vol. 30, No. 1, März 2001.
- [4] Jim Melton u. a.: SQL/MED – A Status Report. In: ACM SIGMOD Record, Vol. 31, No. 3, September 2002.
- [5] Jon Ellis u. a.: JDBC™ 3.0 Specification, Final Release Sun Microsystems, Inc., Oktober 2001.
- [6] Bill Joy u. a.: The Java™ Language Specification, Second Edition Addison-Wesley, Juni 2000.
- [7] Andrew Eisenberg u. a.: SQL/XML and the SQLX Informal Group of Companies. In: ACM SIGMOD Record, Vol. 30, No. 3, September 2001.
- [8] Andrew Eisenberg u. a.: SQL/XML is Making Good Progress. In: ACM SIGMOD Record, Vol. 31, No. 2, Juni 2002.

### Deutschsprachige Literatur zu SQL:1999/SQL:2003

W. Panny; A. Taudes: Einführung in den Sprachkern von SQL-99, Springer, 2000.  
 G. Saake; W. Sattler: Datenbanken & Java (JDBC, SQLJ und ODMG), dpunkt, 2000.  
 C. Türker: SQL:1999 & SQL:2003 (Objekt-relationales SQL, SQLJ & SQL/XML), dpunkt, 2003.  
 D. Petkovic: SQL objektorientiert, Addison-Wesley, 2003.