

Die Datenbanksprache SQL2003: relationale Bestandteile (Implementierungsphase)

5.1 Einführung in SQL	197
5.2 Datentypen, Operatoren, Funktionen und Systemvariablen	200
5.3 Die Datendefinitionssprache (DDL, Data Definition Language)	212
5.4 Die Datenmanipulationssprache (DML, Data Manipulation Language)	241
5.5 Die Datenabfragesprache (DQL, Data Query Language)	244
5.6 Die Datenadministrationssprache (DAL, Data Administration Language)	279
5.7 SQL und die Objekte der Relationalen Algebra ..	281
5.8 Data-Dictionarys	283
Zusammenfassung	286
Weiterführende Literatur	286
Übungsaufgaben	287

» In diesem Kapitel bewegen wir uns in unserem Vorgehensmodell weiter zur Implementierungsphase mittels SQL, einer gerade für die praktische Anwendung von Datenbanksystemen enorm wichtigen Sprache, ohne die kein Datenbankentwickler auskommt. SQL wird in allen Datenbankanwendungen verwendet, die einen relationalen Kern haben – unabhängig davon, in welcher Programmiersprache sie letztendlich geschrieben sind. Wir haben SQL gemäß dem aktuellen SQL2003-Standard dargestellt, allerdings nur in Ausschnitten, da eine vollständige Übersicht den Rahmen dieses Buchs bei weitem sprengen würde. So hat der aktuelle SQL2003-Standard schon über 3000 Seiten, während der SQL1-Standard von 1989 noch mit 120 Seiten auskam. Das Verhältnis der Computerindustrie zu diesem Standard ist zwiespältig. Die einzelnen Anbieter sind im Allgemeinen zwar in großen Bereichen zum Standard kompatibel, aber es gibt immer noch Standardfunktionalitäten, die seit Jahren in der Praxis nicht umgesetzt wurden. Andererseits werden vielfach Erweiterungen angeboten, die zwar nicht standardisiert sind, aber in der Praxis benötigt werden. Die Implementierungen der einzelnen Hersteller unterscheiden sich allerdings nicht so stark, dass es große Schwierigkeiten bereitet, von einem System auf ein anderes umzusteigen. Da wir im praktischen Teil dieses Buchs die Datenbanksysteme Oracle und MySQL im Vergleich verwenden, beziehen sich die Erweiterungen über den SQL-Standard hinaus auf diese beiden Datenbanksysteme. Das trifft insbesondere auf Datentypen, gespeicherte Routinen und objektrelationale Erweiterungen zu. Herstellerspezifische Erweiterungen werden als solche markiert, wir werden uns aber soweit wie möglich an den Standard halten, um den Umstieg auf eine andere Datenbank zu erleichtern. Getestet wurden die einzelnen Anweisungen unter Oracle und MySQL. Die jeweils herstellerspezifischen Besonderheiten werden in eigenen Kapiteln herausgearbeitet.

Inhalt dieses Kapitels sind Grundlagen von SQL, die man zur Datendefinition, Datenmanipulation und zum Suchen benötigt. Mit den objektrelationalen Erweiterungen und der Anbindung von SQL-Datenbanken an Java (SQLJ und JDBC) beschäftigen wir uns in Kapitel 6, gespeicherte Routinen und Datenbanktrigger sind das Thema in Kapitel 7, Transaktionen werden ausführlich in Kapitel 8 beschrieben. Wegen der großen praktischen Relevanz des SQL-Themas ist hier die Bedeutung der umfangreichen Übungsaufgaben hervorzuheben, die neben den auch in den anderen Kapiteln angebotenen Multiple-Choice-Fragen noch einen eigenen SQL-Trainer beinhalten. Mit dem SQL-Trainer können zufällig ausgewählte SQL-Aufgaben aus verschiedenen Datenbankschemata gelöst werden. Die eigene Lösung wird mit dem Ergebnis der Musteranfrage verglichen und bewertet. Den SQL-Trainer finden Sie ebenso wie den Multiple-Choice-Test über die CWS-Seite des Verlages. «



Ziele

Nach Durcharbeiten dieses Kapitels und dem Lösen der Übungsaufgaben werden Sie in der Lage sein,

- das SQL-Datenmodell mit seinen Basisdatentypen und Typkonstruktoren anzuwenden,
- Tabellen, Views, Indizes, Sequenzen und andere Datenbankobjekte anzulegen, zu ändern und zu löschen,
- Daten in Tabellen anzulegen, zu ändern und zu löschen,
- den Unterschied zwischen virtuellen und materialisierten Views zu verstehen,
- einfache und komplexe Anfragen in SQL zu formulieren, unter anderem Unterabfragen, auch mit WITH-Klausel, Abfragen mit verschiedenen Join-Operatoren und rekursive Anfragen,
- den Zusammenhang zwischen SQL und den Objekten der relationalen Algebra zu verstehen,
- Datenintegrität unter SQL zu verstehen und Integritätsprüfungen durchzuführen,
- Benutzerrechte zu verwalten,
- das Data Dictionary von Oracle und MySQL zu benutzen sowie
- Unterschiede in der SQL-Syntax beim Standard 2003 und den Oracle-SQL- und MySQL-Dialekt zu beurteilen.

5.1 Einführung in SQL

5.1.1 Historisches

1970 schrieb E.F. Codd seinen grundlegenden Artikel „A relational model of data for large data banks“ in den Comm. ACM¹, der als Grundlage für das relationale Datenmodell und die relationale Algebra angesehen werden kann. Damals gab es schon erste Überlegungen, wie man diese Ideen und Modelle eines relationalen Systems implementieren könnte. Vorläufer der Sprache SQL war die Sprache SEQUEL, die bei der IBM im Rahmen des Projekts System R entwickelt wurde. 1981 gelangte diese Sprache mit SQL/Data Systems erstmalig auf den Markt. Schon bald folgten andere Datenbanksysteme, u.a. DB2 und Oracle. Der Ansatz der IBM setzte sich wegen seiner Mächtigkeit letztendlich durch. 1980 begannen die Arbeiten am ersten Standard, bekannt unter SQL1, der 1986 von der ANSI² und 1989 von der ISO³-Behörde verabschiedet wurde. SQL hat sich im Laufe der Zeit zu einer sehr umfangreichen Datenbanksprache entwickelt, die in den verschiedenen ANSI-SQL-Standards SQL1 (1986), SQL2 (1991-1992) und SQL3 (1999-2003) beschrieben wird. Da sich die Veröffentlichung der 3. Version von 1999 bis 2003 hinzugezogen hat, verwenden wir die Schreibweise SQL2003, um zu verdeutlichen, dass wir uns immer auf die letzte Version beziehen.

¹ vgl. [Codd 1970]

² ANSI – American National Standards Institute: <http://www.ansi.org/>, 10.12.2006

³ ISO – International Organisation for Standardization. Ein Verbund von nationalen Standardisierungsorganisationen aus 157 Ländern: <http://www.iso.org>, 10.12. 2006

5.1.1.1 Die Sprachbestandteile von SQL1

- Datendefinitionssprache (Data Definition Language, DDL) mit den Anweisungen CREATE TABLE, CREATE VIEW etc.
- Sprache zur Definition von Speicherstrukturen, Benutzerrechten und anderen Verwaltungsaufgaben (Data Storage Definition Language, DSDL, und Data Administration Language, DAL) mit den Anweisungen COMMIT, ROLLBACK, GRANT und REVOKE
- Datenmanipulationssprache (Data Manipulation Language, DML) mit den Anweisungen INSERT, UPDATE und DELETE
- Datenabfragesprache (Data Query Language, DQL) mit Varianten der SELECT-Anweisung

5.1.1.2 Die Sprachbestandteile von SQL2

1991 wurde ein weiterer SQL-Standard, SQL2, verabschiedet, der z.B. die Domänen, einige neue Datentypen (BLOBS, VARCHAR, DATE, TIME, TIMESTAMP, BOOLEAN) sowie die Veränderungen von Tabellen mit ALTER TABLE und Transaktionen zum Inhalt hat. Auch die Möglichkeit, Integritätsprüfungen vorzunehmen, gab es schon. Dazu gehören die Vereinbarung von CHECK-Klauseln, um Gültigkeitsprüfungen für einzuftigende Werte vorzunehmen und das Festlegen von Primärschlüsseln, Fremdschlüsseln und Zweitschlüsseln. In der DQL-Sprache wurden die verschiedenen JOIN-Operatoren (INNER-JOIN, OUTER-JOIN) ergänzt. Die meisten Datenbanksysteme unterstützen bis heute diesen Standard nicht vollständig, andererseits gehen sie in vielen Punkten weit über den Standard hinaus.

5.1.1.3 Der SQL2003-Standard

Neuerungen in SQL2003 sind insbesondere:

- Neue Datentypen: BIGINT und XML-Typ (als völlig neuer Bestandteil)
- Generierte Spalten und Sequenzgeneratoren (CREATE SEQUENCE)
- Generierte Tabellen mit CREATE TABLE AS SELECT und CREATE TABLE LIKE
- Tabellenwertige Funktionen, d.h. persistent gespeicherte Funktionen, die als Rückgabewert eine Tabelle haben
- Der MERGE-Operator, um Tabelleninhalte zu verschmelzen
- SQLJ und JDBC 3.0 (teilweise schon in SQL1999)
- Objektrelationale Erweiterungen (benutzerdefinierte Typen schon in SQL1999) und insbesondere der Kollektionstyp MULTISSET in SQL 2003
- Definition von Stored Procedures und Datenbanktriggern (PSM, Persistent Storage Modules) schon in SQL1999
- SUBQUERYs in CHECK-Bedingungen und ASSERTIONS (SQL1999)

5.1.2 Das SQL-Datenmodell und die Implementierungsphase

Verglichen mit dem relationalen Datenmodell aus Kapitel 4 gibt es bei SQL nur eine wesentliche Erweiterung: Der Typkonstruktur SET wird durch den Typkonstruktur MULTISSET ausgetauscht. Damit ist nicht der neue Typkonstruktur unter SQL2003 gemeint, der als objektrelationale Erweiterung in Kapitel 6 beschrieben wird, sondern einfach nur die Tatsache, dass in SQL-Tabellen auch Duplikate erlaubt sind. Nach Türker lässt sich das SQL-Datenmodell dann gegenüber der ► Abbildung 4.2 auf Seite 129 folgendermaßen grafisch darstellen:

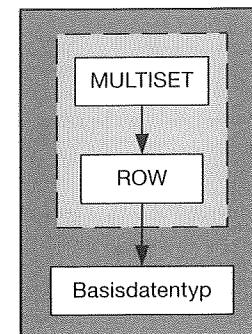


Abbildung 5.1: Relationenmodell als Datenmodell⁴

Die Tabellen sind in dieser Grafik als MULTISSET(ROW(Basisdatentyp)) definiert. Die einzigen Typkonstruktoren sind also ROW und MULTISSET. Die Typkonstruktionsregel MULTISSET(ROW(Basisdatentyp)) gibt an, dass man MULTISSET auf ROW anwenden kann und ROW auf die Basisdatentypen. Andere Typkonstruktionsregeln sind nicht zulässig. Die Basisdatentypen werden detailliert in Kapitel 5.2 vorgestellt. Der Einstiegspunkt in die Datenbank ist die gestrichelte Linie. Objektrelationale Erweiterungen mit dem dazugehörigen erweiterten Datenmodell werden in diesem Kapitel nicht beschrieben, sondern sind in ein eigenes Kapitel 6 ausgelagert, um den Anfänger erst mal mit den Grundzügen von SQL in der herkömmlichen Weise vertraut zu machen.

Eingebettet in unser Vorgehensmodell zur Erstellung einer Datenbank aus Kapitel 2 befinden wir uns jetzt in der Implementierungsphase. Das aus dem konzeptionellen Modell aus Kapitel 3 abgeleitete Datenbankschema (Designphase) wird auf SQL-Befehle abgebildet und unter anderem mittels CREATE-TABLE-Anweisungen in der Datenbank implementiert. Wir stellen im folgenden Kapitel einige Konventionen zur Schreibweise von SQL-Befehlen vor.

⁴ vgl. [Türker et al. 2006, S. 55]

5.1.3 Konventionen zur Schreibweise von SQL-Befehlen

Tabelle 5.1

Verwendete Vereinbarungen bei der Definition von SQL-Anweisungen	
Symbol	Bedeutung
GROSSBUCHSTABEN	Schlüsselwörter der Sprache
<Text>	Kursiver Text in spitzen Klammern bezeichnet Variablen, die durch Definitionen ersetzt werden müssen.
Kursiver Text	Kursiver Text bezeichnet ein Terminal, dazu gehören auch Bezeichnungen, wie „Tabellename“, die vom Benutzer selbst bestimmt werden können und für die es keine weitere Definition gibt.
::=	Definitionsanweisung x ::= y bedeutet: x wird durch y definiert.
	Alternative x y bedeutet: x oder y muss angegeben werden. steht in einer Liste von Elementen, die in geschweiften oder eckigen Klammern eingeschlossen ist.
[]	Option Die eckigen Klammern stehen für einen optionalen Ausdruck.
{}	Gruppierung Die geschweiften Klammern stehen für einen Ausdruck, der enthalten sein muss.
...	Wiederholung Der zuletzt geschriebene Ausdruck kann, muss aber nicht, beliebig oft wiederholt werden.

SQL selber unterscheidet keine GROSS/Kleinschreibung. Trotzdem werden wir uns der besseren Lesbarkeit halber an die Konvention halten, Schlüsselwörter in Großbuchstaben zu schreiben und Tabellen- und Sapltennamen etc. in Kleinbuchstaben.

5.2 Datentypen, Operatoren, Funktionen und Systemvariablen

5.2.1 Datentypen

Grundlage der Datendefinitionssprache sind die Datentypen, die vom ANSI-Komitee schon zum größten Teil im SQL2-Standard festgelegt wurden. Leider sind die Datentypen bei den einzelnen Herstellern sehr verschieden implementiert und weichen zum Teil stark vom Standard ab. Einen (nicht vollständigen) Vergleich der wichtigsten Typen finden Sie in der folgenden Tabelle:

Tabelle 5.2

(Nicht vollständiger) Vergleich der wichtigsten Datentypen

Datentyp	Beschreibung	ANSI	Oracle	MySQL
SMALLINT	Ganze Zahl zwischen -32.767 und 32767	x	x	x
BIGINT	Ganze Zahl zwischen -2 ⁶³ und 2 ⁶³ -1	x		x
INTEGER	Ganze Zahl zwischen -2.147.483.647 und 2.147.483.647	x	x	x
INT				
FLOAT	Gleitkommazahl zwischen -4.10E79 und 4.10E79	x	x	x
REAL	7.210E75			
DOUBLE	Gleitkommazahl mit 8 Byte	x	x	x
DECIMAL(p,q)	Festkommazahl mit p Ziffern und q Nachkommastellen, p zwischen 1 und 38, q zwischen -38 und 38	x	x	x
NUMERIC(p,q)				
CHARACTER(n)	Wort mit fester Zeichenlänge von n Zeichen, n ≤ 2000 Zeichen	x	x	x
CHAR(n)				
VARCHAR(n)	Wort mit variabler Zeichenlänge von n Zeichen	x	x	x
VARCHAR2(n)				
RAW	Binärer Datentyp bis zu 2000 Byte		x	
LONG RAW	Binärer Datentyp bis zu 2 GByte (z.B. für Fotos, Grafiken und andere Binärdaten)		x	
DATE	Datumsfelder, teilweise mit Uhrzeit	x	x	x
DATETIME	Datumsfelder mit Uhrzeit			x
TIME	Uhrzeiten	x		x
TIMESTAMP	Zeitstempel mit Datum und Uhrzeit	x	x	x
ROWID	Hexadezimale Adresse des Datensatzes		x	(x) ⁶
INTERVAL	Zeitintervalle	x		x
BLOB	Für Binärdaten wie Fotos, Grafiken und Ton	x	x	x
CLOB	Für lange Textobjekte		x	
BFILE	Zeiger auf eine Datei, in der ein Text oder Bildobjekt gespeichert ist		x	

⁵ n ist je nach Hersteller unterschiedlich ausgelegt, bei Oracle ist n = 4000, bei MySQL n= 254.

⁶ Die Verwendung der ROWID unter MySQL weist einige Besonderheiten auf (vgl. Abschnitt 5.4.5).

(Nicht vollständiger) Vergleich der wichtigsten Datentypen (Fortsetzung)					
Datentyp	Beschreibung	ANSI	Oracle	MySQL	
TEXT	Für lange Textobjekte		x		
TEXT					
LONGBLOB					
LONGTEXT					
BOOLEAN	Boolescher Datentyp	x	(x)		
XML	Für XML-Dokumente	x	x ⁷		
ENUM ⁸ (Wert1, Wert2...)	Enumeration vom Typ String, die nur genau einen der bis zu 65535 angegebenen Werte Wert1 ... annehmen kann (statischer Wertebereich)		x		
SET (Wert1, Wert2...)	Menge von maximal 64 String-Objekten; SET besteht aus einer Teilmenge der Objekte der Werteliste Wert1		x		

5.2.1.1 Unterschied zwischen CHAR und VARCHAR

Bei CHAR oder CHARACTER-Feldern wird die vereinbarte Textlänge vollständig reserviert, bei VARCHAR (oder VARCHAR2) nur der tatsächlich benötigte Speicherplatz. VARCHAR wird vor allem bei langen Zeichenketten mit unterschiedlichen Längen verwendet. Problematisch ist der Vergleich der beiden Datentypen, da die CHAR-Spalten mit Leerzeichen aufgefüllt werden. Im Beispiel unten (HEXE) würde der Vergleich auf Gleichheit FALSE liefern, da die vier anhängenden Leerzeichen mit verglichen werden. Eine Möglichkeit, damit umzugehen, stellt die Verwendung der RTRIM-Funktion dar (vgl. Abschnitt 5.2.3) oder die ausschließliche Verwendung von VARCHAR.

VARCHAR(8)	H E X E
CHAR(8)	H E X E

Der Oracle-Datentyp VARCHAR2 wird anstelle von VARCHAR verwendet und hat eine Länge von 4000 Byte. MySQL sieht die Standarddatentypen für numerische Werte und Textdatentypen vor. Oracle verwendet statt der ANSI-Typen DECIMAL, NUMERIC und FLOAT intern immer den NUMBER-Datentyp. Die Standarddatentypen aus ANSI-SQL sind zwar verwendbar, intern wird aber vom Oracle-DBMS das NUMBER-Format bzw. VARCHAR2-Format eingesetzt.

5.2.1.2 Umwandlung von ANSI-Datentypen in interne Oracle-Datentypen

Tabelle 5.3	
Umwandlung von ANSI-Datentypen in interne Oracle-Datentypen	
ANSI-SQL-Datentyp	Oracle-Datentyp
NUMERIC(p,s)	NUMBER(p,s)
DECIMAL(p,s)	
INTEGER	NUMBER(38)
SMALLINT	
BIGINT	
FLOAT	NUMBER
CHARACTER	CHAR
VARCHAR	VARCHAR2

5.2.1.3 Auswirkung von p und q auf NUMERIC-, DECIMAL- und NUMBER-Datentypen

p steht für „precision“ und meint die Gesamtzahl der Stellen ohne das Dezimaltrennzeichen. q ist die Anzahl von Zeichen nach dem Dezimaltrennzeichen. Fehlt der Wert q, handelt es sich um eine ganzzahlige Variable. Fehlen beide Angaben, ist eine numerische Variable ohne genaue Stellenbegrenzung gemeint. q kann auch negative Werte annehmen. In diesem Fall wird auf Zehnerstellen vor dem Komma gerundet.

Tabelle 5.4		
Anzeige numerischer Datentypen		
Datensatz	Datentyp	Auswirkung
7456123.89	NUMERIC	7456123.89
7456123.89	NUMERIC(9)	7456124
7456123.89	NUMERIC(9,2)	7456123.89
7456123.89	NUMERIC(9,1)	7456123.9
7456123.89	NUMERIC(6)	nicht speicherbar
7456123.89	NUMERIC(7,-2)	7456100
7456123.89	NUMERIC(7, -1)	7456120

⁷ Der Datentyp XML heißt unter Oracle XMLType.

⁸ vgl. Abschnitt 6.1.10, dort werden SET und ENUM unter MySQL noch näher beschrieben.

5.2.1.4 Bestandteile der DATE-Typen

DATE ist ein Datentyp für formatierte Datumsfelder. Für jeden DATE-Datentyp in SQL und MySQL werden gespeichert:

- das Jahr,
- der Monat und
- der Tag.

Der DATE-Datentyp von Oracle umfasst zusätzlich auch die Uhrzeit mit

- Stunden,
- Minuten und
- Sekunden

und entspricht damit den ANSI- und MySQL-Datentypen TIMESTAMP und DATETIME, die mittlerweile von Oracle auch unterstützt werden.

5.2.1.5 Boolescher Datentyp

BOOLEAN ist im SQL-Standard von 1999 enthalten, aber noch selten implementiert. In SQL ist es ein dreiwertiger Datentyp (TRUE, FALSE, UNKNOWN/NULL) und bei MySQL nur ein zweiwertiger. Es ist dort ein Synonym für TINYINT(1), dessen Wert 0 als FALSE interpretiert wird. Alle anderen Zustände werden als TRUE interpretiert. Den booleschen Datentyp gibt es bei Oracle nur für PL/SQL (vgl. Abschnitt 7.1).

5.2.1.6 Multimediatentypen

RAW, LONG RAW in Oracle Mit dem Datentyp RAW können Binärdaten bis zu einer Länge von 255 Byte gespeichert werden, bei LONG RAW sind es bis zu 2 Gbyte Daten. Diese Datentypen werden genutzt, um z.B. Grafiken, Tondokumente oder lange Texte zu speichern. In Anlehnung an SQL2003 werden sie durch die BLOB-Datentypen ersetzt.

BLOB Mit diesem Datentyp können entsprechend dem SQL-Standard binäre Objekte (Oracle bis zu 4 Gbyte) für Grafiken oder Tonaufzeichnungen in der Datenbank gespeichert werden. Außerdem stehen einige Methoden zur Verfügung, mit denen diese Objekte auf dem Server manipuliert werden können.

CLOB und NCLOB in Oracle CLOB ist eine Modifikation des Typs BLOB, er wird für große Textobjekte mit den entsprechenden Methoden zur Verfügung gestellt. NCLOB entspricht dem CLOB mit nationalem Zeichensatz.

BLOB, MEDIUMBLOB, LONGBLOB, TEXT und LONGTEXT in MySQL BLOB entspricht unter MySQL dem ANSI-Standard, MEDIUMBLOB und LONGBLOB sind MySQL-spezifisch, benötigen mehr Speicherplatz und sind daher für größere Objekte vorgesehen: BLOB kann maximal 64 Kbyte aufnehmen, MEDIUMBLOB 16 Mbyte und LONGBLOB 4 Gbyte. TEXT und LONGTEXT sind wieder MySQL-spezifisch und ersetzen den Typ CLOB für lange Texte (ANSI-Standard).

Benutzerdefinierte Datentypen (SQL1999) sowie die Verarbeitung von BLOBS und Kollektionen sind Inhalt von Kapitel 6.

5.2.2 Operatoren

Operatoren werden benutzt, um Daten zu verknüpfen und ein Resultat auszugeben. Sie entsprechen dem ANSI/SQL2003-Standard.

Tabelle 5.5

Operatoren		
Operator	Beschreibung	Beispiel
+ ⁹	Positiver oder negativer Ausdruck	SELECT * FROM Teile WHERE - Bestand < 0;
*	Multiplikation	UPDATE Angestellte SET Gehalt = Gehalt *
/	Division	1.1;
+ ¹⁰	Addition	SELECT Gehalt - Abzuege FROM Angestellte;
-	Subtraktion	
	Konkatenation	SELECT Vorname ' ' Nachname FROM Angestellte;

Für die Rechenoperatoren ist die allseits bekannte Funktionalität implementiert. Da NULL-Werte von Spalten zulässig sind, muss lediglich berücksichtigt werden, wie das Ergebnis aussieht, wenn einer der Operanden NULL ist. Ist das der Fall beim positiven oder negativen Vorzeichen, bei der Addition, Subtraktion, Multiplikation oder Division, so ist das Ergebnis auch NULL. Beachten Sie beim Umgang mit leeren Spalten die unterschiedlichen Darstellungen. Bei Oracle wird die Spalte leer angezeigt und bei MySQL mit der Kennung NULL. Die Operatorschreibweise „||“ für die CONCAT-Funktion (vgl. Abschnitt 5.2.3) kann in MySQL aber nur im SQL-Modus „ANSI“ ausgeführt werden (vgl. SET SQL_MODE-Anweisung im Abschnitt 5.3.2).

Vergleichsoperatoren werden in logischen Ausdrücken benutzt, z.B. in der CHECK-Bedingung eines CONSTRAINTS (vgl. Abschnitt 5.3.2) oder in der WHERE-Bedingung einer Anfrage (vgl. Abschnitt 5.5). Im Detail und mit Beispielen werden die Operatoren im Abschnitt 5.5.4 vorgestellt. Hier sei aber bereits darauf verwiesen, dass in SQL, bei Oracle und MySQL eine dreiwertige Logik zugrunde liegt und bei jedem Operator und jeder Funktion explizit geschaut werden muss, wie das Ergebnis aussieht, wenn einer der Ausdrücke NULL ist (vgl. Abschnitt 5.5.2).

In den folgenden Tabellen stellen wir die Vergleichsoperatoren, die SQL-Operatoren sowie die logischen Operatoren vor.

9 Unter Oracle-SQL und dem Standard-SQL ist auch die Addition und Subtraktion von Datumsfeldern mit einem Integer-Wert als Anzahl von Tagen zugelassen. Unter MySQL müssen dazu Funktionen wie z.B. ADDDATE(date, zahl) benutzt werden.

10 Ebenda

Tabelle 5.6

Vergleichsoperatoren	
Operator	Bedeutung
=	Gleich
>	Größer
>=	Größer gleich
<	Kleiner
<=	Kleiner gleich
<> und !=	Ungleich

Tabelle 5.7

SQL-Operatoren	
Operator	Bedeutung
[NOT] BETWEEN Untergrenze AND Obergrenze	[NOT] TRUE, wenn ein Wert zwischen zwei Werten liegt, einschließlich der Grenzen
[NOT] IN (Wert1, Wert2 ...)	[NOT] TRUE, wenn Übereinstimmung mit irgendeinem Wert der Liste existiert ¹¹
[NOT] LIKE	[NOT] TRUE, wenn Übereinstimmung mit einem Zeichenmuster existiert; Wildcards: "%" und "_"
IS [NOT] NULL	[NOT] TRUE, wenn Spalte, Variable leer sind

Tabelle 5.8

Logische Operatoren	
Operator	Bedeutung
AND	Wenn beide Teilbedingungen TRUE sind, ist auch das Ergebnis TRUE.
OR	Wenn mindestens eine von beiden Teilbedingungen TRUE ist, ist auch das Ergebnis TRUE.
NOT	Negiert das Ergebnis der Bedingung
XOR	Logisches, exklusives Oder: Genau eine von zwei Teilbedingungen muss TRUE sein, damit das Ergebnis TRUE ist (MySQL-spezifisch).

¹¹ Vom IN-Operator gibt es auch eine Variante mit einer Unteranfrage (vgl. Abschnitt 5.5.9).

5.2.3 Funktionen

Die Funktionen werden in Single-Row-Funktionen und Multiple-Row-Funktionen (Gruppenfunktionen) unterschieden. Die Gruppenfunktionen fassen Werte aus unterschiedlichen Datensätzen zu einem Wert zusammen und gehören schon zum SQL1-Standard. Single-Row-Funktionen liefern aus einem Ausdruck einer Zeile einen Wert zurück. Diese Funktionen sind zu einem großen Teil herstellerabhängig, ein Teil wurde in den SQL2-Standard aufgenommen. Es folgt eine Auswahl an Single-Row-Funktionen, die häufig benutzt werden und unter Oracle und MySQL übereinstimmen.¹²

Tabelle 5.9

Eine Auswahl an Single-Row-Funktionen auf Zeichenketten (Oracle und MySQL)

Funktion	Beschreibung	Beispiel
LOWER (String)	Konvertiert String in Kleinbuchstaben	SELECT LOWER (Nachname) FROM Angestellte;
UPPER (String)	Konvertiert String in Großbuchstaben	SELECT UPPER(Nachname) FROM Angestellte;
LENGTH ¹³ (String)	Liefert die Anzahl der Zeichen des Strings	SELECT LENGTH (Nachname) FROM Angestellte;
SUBSTR (String, m [n])	Liefert eine Zeichenkette der Länge n ab der Position m	SELECT SUBSTR(Nachname, 2,2) FROM Angestellte;
CONCAT (String1, String2 ...)	Hängt zwei Strings aneinander	SELECT CONCAT('Hugo ', 'Schmidt') FROM DUAL;
ASCII	Wandelt ein einzelnes Zeichen in ASCII-Code um	SELECT ASCII('F') FROM DUAL;
CHR	Wandelt ASCII-Code in Zeichen	SELECT CHR(70) FROM DUAL;
TRIM	Entfernt Suffixe und Präfixe	SELECT TRIM('H' FROM 'Hugo') FROM DUAL;
LPAD RPAD	Füllt Zeichenketten auf eine bestimmte Länge von links bzw. rechts mit Füllzeichen auf	SELECT LPAD('Hugo Müller', 25, 'x') FROM DUAL;

Die Tabelle „DUAL“ ist Oracle-spezifisch. Sie enthält nur einen Datensatz und ist nicht änderbar. Sie dient lediglich als Hilfsmittel, damit man Anfragen mit Funktionsaufrufen schreiben kann, die auch nur genau einmal ausgeführt werden. In MySQL gibt es für diesen Fall eine besondere Kurzschreibweise der SELECT-Anfrage ohne die sonst notwendige FROM-Klausel. Zudem wird diese Kurzform noch zur Anzeige von

¹² In der Oracle-Literatur (z. B. [ORACLE SQL 2006]) bzw. der MySQL-Literatur [MySQL 2006] finden Sie eine vollständige Auflistung der Funktionen.

¹³ Diese Funktion heißt im ANSI-Standard SQL2 CHAR_LENGTH.

Meldungen aus gespeicherten Prozeduren (vgl. Abschnitt 7.2) heraus verwendet. In MySQL sieht eine entsprechende Anfrage folgendermaßen aus:

```
SELECT TRIM('H' FROM 'Hugo');
SELECT LPAD('Hugo Müller', 25, 'x');
```

5.2.3.1 Numerische und mathematische Single-Row-Funktionen

Tabelle 5.10

Eine Auswahl an numerischen und mathematischen Single-Row-Funktionen (Oracle und MySQL)		
Funktion	Beschreibung	Beispiel
ROUND (Ausdruck, n)	Rundet den Ausdruck auf n Dezimalstellen	SELECT ROUND(Gehalt, 2) FROM Angestellte;
MOD (m,n)	Liefert den ganzzahligen Rest von m dividiert durch n	SELECT MOD (Gehalt, 2) FROM Angestellte;
TRUNC (Ausdruck, n)	Schneidet den Ausdruck nach n Dezimalstellen ab	
NVL ¹⁴ (Spalte, Wert)	Ersetzt einen NULL-Wert in der Spalte durch den angegebenen Wert, sonst wird der Wert der Spalte zurückgegeben	SELECT NVL(Gehalt, 0) FROM Angestellte;
IFNULL (Spalte, Wert)		SELECT IFNULL(Gehalt, 0) FROM Angestellte;
ABS, COS, SIN, LN, LOG, SIGN, POWER, SINH, SQRT, TAN	Verschiedene mathematische Funktionen, siehe Hersteller-Originalliteratur	

5.2.3.2 Single-Row-Datumsfunktionen

Tabelle 5.11

Eine Auswahl an Single-Row-Datumsfunktionen (Oracle)		
Funktion	Beschreibung	Beispiel
MONTHS_BETWEEN	Berechnet die Datumsdifferenz in Monaten	MONTHS_BETWEEN('15.01.2000', '01.12.1999') ergibt: 1,4516129
ADD_MONTH	Addiert Monate zum Datum	ADD_MONTHS('01.09.1999', 6) ergibt '01.03.00'
NEXT_DAY	Erster Wochentag nach dem Datum	NEXT_DAY('01.09.95', 'FRIDAY') ergibt '02.09.95'

¹⁴ NVL ist Oracle-spezifisch, IFNULL ist die entsprechende Funktion unter MySQL.

Eine Auswahl an Single-Row-Datumsfunktionen (Oracle) (Fortsetzung)

Funktion	Beschreibung	Beispiel
LAST_DAY	Letzter Tag des Monats	LAST_DAY('01.09.99') ergibt '30.09.99'
ROUND	Rundet auf Monate, Jahre, Tage usw.	ROUND ('25.05.95', 'MONTH') ergibt '01-JUN-95'
TRUNC	Schneidet Monate, Jahre, Tage usw. ab	TRUNC ('25-MAY-95', 'MONTH') ergibt '01-MAY-95'

Die MySQL-Datumsfunktionen sind sehr umfangreich und können nur zu einem Bruchteil angegeben werden. Es wird wieder auf die Originalliteratur [MySQL 2006] bzw. [Buchmann 2005] verwiesen.

Tabelle 5.12

Eine Auswahl an Single-Row-Datumsfunktionen (MySQL)

Funktion	Beschreibung	Beispiel
ADDDATE (datum, interval)	Addiert Tage oder ein Intervall zum Ausdruck	ADDDATE('01.09.2006', 6)
MONTH (datum)	Extrahiert die Monatszahl aus einem Datum	MONTH('01.09.2006')
YEAR (datum)	Extrahiert die Jahreszahl aus einem Datum	YEAR('01.09.2006')
DATEDIFF(datum1, datum2)	Differenz in Tagen	DATEDIFF('01.09.06', 12.12.2007')
CURRENT_TIME() / NOW()	Gibt das Systemdatum aus	CURRENT_TIME()

5.2.3.3 Konvertierungsfunktionen zur Typumwandlung (Oracle, MySQL)

Unter Oracle gibt es Funktionen wie TO_CHAR, TO_NUMERIC, TO_DATE und TO_VARCHAR, die einen Ausdruck in den entsprechenden Typ umwandeln. Unter MySQL heißen diese Funktionen CHAR, NUMERIC, DATE und VARCHAR. Daneben gibt es in beiden Implementierungen die SQL-konforme Funktion CAST¹⁵ (Ausdruck AS-Typ), die einen Ausdruck in den vorgegebenen Typ umwandelt. Die obigen Funktionen können beliebig ineinander geschachtelt werden.

¹⁵ Unter MySQL ist CAST nur für die Zeichentypumwandlung von einer Textvariablen in eine andere unter einem anderen Zeichensatz vorgesehen.

Beispiele

```
SELECT CONCAT(SUBSTR(Vorname,3), UPPER(Nachname))
FROM Angestellte;
```

```
SELECT Ang_Nr, CAST(Gehalt AS VARCHAR(50)) FROM Angestellte;
```

5.2.3.4 SQL-Gruppenfunktionen

Gruppenfunktionen (oder auch Aggregatfunktionen) fassen mehrere Werte aus einer Spalte zu einem Wert zusammen (vgl. Abschnitt 5.5.7).

Tabelle 5.13

SQL-Gruppenfunktionen	
Funktion	Bedeutung
COUNT	Anzahl der Werte in einer Spalte
SUM	Summe der Werte in einer Spalte
AVG	Mittelwert der Spalte
MAX	Größter Wert der Spalte
MIN	Kleinster Wert der Spalte

5.2.4 Systemvariablen und Wildcards**5.2.4.1 Systemvariablen**

Schon im SQL2-Standard sind einige Systemvariablen vorgesehen, die zur Laufzeit abgefragt werden können und allen Benutzern zur Verfügung stehen.

Tabelle 5.14

SQL-Systemvariablen	
Variable	Beschreibung
CURRENT_DATE	Systemdatum
CURRENT_TIME	Aktuelle Zeit
CURRENT_USER	Eingeloggter Benutzer
SESSION_USER	Benutzer, der eine Session gestartet hat
SYSTEM_USER	Systemverwalter

Systemvariablen werden in Abfragen oder z.B. als DEFAULT-Wert bei der Tabellendefinition verwendet. Diese Systemvariablen sind leider wieder nicht einheitlich bei den jeweiligen Herstellern implementiert. Unter MySQL sind die obigen Variablen als Funktionen implementiert, daher die Aufrufe mit den Klammern: CURRENT_DATE(), CURRENT_TIME(), CURRENT_USER(), SESSION_USER(), SYSTEM_USER(). Die Funktion SYSDATE() liefert analog zur Oracle-Funktion SYSDATE das Tagesdatum mit Uhrzeit. Unter Oracle sieht diese Tabelle ganz anders aus. Allerdings sind in den neueren Versionen von Oracle (ab Oracle 9) auch die ANSI-Systemvariablen teilweise implementiert.

Tabelle 5.15

Oracle-Systemvariablen	
Oracle- Variable	Bedeutung
SYSDATE	Aktuelles Datum mit Uhrzeit
DUAL	Dummy-Tabelle
USER	Eingeloggter Benutzer
UID	Interne Kennung des eingeloggten Benutzers

5.2.4.2 Wildcards

Wildcards werden in Schablonen verwendet, um Textmuster zu suchen. Wir werden sie in Checkbedingungen und Suchbedingungen verwenden (vgl. Abschnitt 5.5.5).

Tabelle 5.16

Wildcards	
Symbol	Beschreibung
%	Steht für eine Zeichenkette mit keinen, einem oder mehreren Zeichen
_	Steht für ein einzelnes Zeichen

5.3 Die Datendefinitionssprache (DDL, Data Definition Language)

In diesem Kapitel erfahren Sie, wie man relationale Objekte definiert, ändert, umbenennst und löscht. Zu den relationalen Objekten, die mit SQL verwaltet werden können, zählen im Wesentlichen die in der folgenden Übersicht aufgeführten.

Tabelle 5.17

Relationale Objekte	
Objekt	Beschreibung
TABLE	Basisrelation als Tabelle
INDEX	Verbessert den Datenzugriff durch den Aufbau von gut zu durchsuchenden separaten Verzeichnissen der Gestalt (Wert, Adresse)
VIEW	Abfrage an die Datenbank, die als virtuelle bzw. materialisierte Tabelle abgelegt ist
SEQUENCE	Erzeugt fortlaufende Nummer

Außer diesen Objekten gibt es noch eine Reihe von herstellerabhängigen Objekten wie USER, SYNONYM, SNAPSHOT, ROLE, PRIVILEGE, TABLESPACE (vgl. Abschnitt 9.7), DATABASE¹⁶ etc., die hier nicht weiter behandelt werden können.

Die einzelnen Objekte kann man

- erzeugen mit der CREATE-Anweisung,
- ändern mit der ALTER-Anweisung,
- löschen mit der DROP-Anweisung und
- umbenennen mit der RENAME-Anweisung.

Bei den einzelnen Anweisungen gibt es jeweils einen Syntaxteil, Anmerkungen und Beispiele. Alle Beschreibungen der einzelnen Anweisungen sind nach diesem Schema aufgebaut.

5.3.1 Die CREATE TABLE-Anweisung

Die CREATE TABLE-Anweisung dient zur Anlage von Tabellen in der Datenbank. Sie ist in ANSI SQL noch recht einfach gehalten, bei den einzelnen Herstellern gibt es umfangreiche Erweiterungen. Die Speicheroptionen vertiefen wir hier nicht weiter, da dies eher in den Bereich der Administration gehört. Die Integritätssicherung wird in den Abschnitten 5.3.2 und 5.3.3 vorgestellt.

¹⁶ vgl. die Herstellerdokumentation von Oracle [ORACLE 2006], [MySQL 2006] oder [Loney 2005]

```

<CREATE TABLE Anweisung> ::= CREATE TABLE Tabellename (<Spaltendefinition>
[ , <Spaltendefinition> ]...
[ , <Tabellenbedingung> ]|...);

<Spaltendefinition> ::= Spaltenname <Datentyp>
[ DEFAULT <Ausdruck> ]
[ <Spaltenbedingung> ]...

<Ausdruck> ::= /<numerischer Ausdruck> |
<alphanumerischer Ausdruck>

<numerischer Ausdruck> ::= / numerische Konstante
| Spaltenname
| <numerischer Ausdruck> <Operator> <numerischer Ausdruck>
| (<numerischer Ausdruck> )
| Funktionsname (<Parameterliste>) |

<alphanumerischer Ausdruck> ::= / alphanumerische Konstante
| Spaltenname
| Systemvariable
| <alphanumerischer Ausdruck> || <alphanumerischer Ausdruck>
| (<alphanumerischer Ausdruck> )
| Funktionsname (<Parameterliste>) |

```

Anmerkungen

- Es empfiehlt sich, Tabellen- und Spaltennamen ohne Umlaute und andere Sonderzeichen zu benennen. Insbesondere sollten Unterstriche anstelle von Bindestrichen und keine Leerzeichen benutzt werden.
- Die Verwendung von Integritätsbedingungen in Form von Spalten- und Tabellenbedingungen wird in den nächsten Kapiteln noch detailliert behandelt.
- Die Datentypen, Operatoren, Vergleichsoperatoren und Single-Row-Funktionen sowie die Systemvariablen können, wie im Abschnitt 5.2 definiert, benutzt werden.
- Der <Ausdruck> kommt hier nur als DEFAULT-Ausdruck und in den Spalten- und Tabellenbedingungen vor. In der DML-Sprache werden wir ihn später auch in anderen Anweisungen benutzen (vgl. Abschnitt 5.4).

Beispiele

```

CREATE TABLE Lager(
LANr      INTEGER      NOT NULL,
LaBez     CHAR(30)    DEFAULT 'Guthe',
PLZ       INTEGER      NOT NULL);

CREATE TABLE Auftragspositionen(
AuftragsNr NUMERIC(10) NOT NULL,
TNr        NUMERIC(10) NOT NULL,
Menge     NUMERIC(10) DEFAULT 1 NOT NULL);

```

5.3.2 Integritätsbedingungen in SQL

Die Integrität einer Datenbasis bezeichnet die semantische Korrektheit der gespeicherten Daten. Unter Integritätssicherung versteht man die Prüfung aller Daten hinsichtlich ihrer Richtigkeit für die gegebene Anwendung. Im Abschnitt 4.2.3 wurden die allgemeinen Konzepte der Integritätssicherung und unterschiedlicher Integritätsarten (statische Bedingungen, dynamische Bedingungen und temporale Bedingungen) vorgestellt. Das Integritätskonzept des SQL-Standards ist seit 1992 im Wesentlichen unverändert – und die anspruchsvolleren Bedingungen sind mehr als 14 Jahre später weder bei Oracle noch bei MySQL und auch bei vielen anderen Datenbanksystemen (z.B. DB2) nicht implementiert. Wir stellen im weiteren kurz vor, was für Integritätsbedingungen denkbar bzw. wünschenswert wären, verglichen mit den allgemeinen Konzepten aus Abschnitt 4.2.3, ordnen dann die des Standards in diesen Kontext ein und erläutern sie. Abschließend wird vorgestellt, welche Konzepte von Oracle und MySQL umgesetzt werden.

Zunächst stellt sich die Frage, warum Integritätsbedingungen in Datenbanksystemen sinnvoll bzw. notwendig sind. Ihr Vorteil ist insbesondere, dass sie einmal zentral in dem Datenbanksystem definiert werden und dann für alle Anwendungen gelten. So wird vermieden, dass in den verschiedenen Anwendungsprogrammen unterschiedliche Prüfungslogik (inkonsistenter, redundanter Code) programmiert wird bzw. die Programmierung in einzelnen Programmen gänzlich vergessen wird. Der aktuell gültige Prüfungscode der einzelnen CONSTRAINTS kann jederzeit in den Data Dictionary-Tabellen nachgesehen werden. Das Problem veralteter Script-Dateien, die irgendwo im Dateisystem gefunden werden (und von denen niemand weiß, ob sie den installierten Prüfungscode enthalten), stellt sich nicht. Aufgrund der deklarativen Formulierung der Bedingungen sind sie leichter lesbar und ihre Bedeutung ist schneller und eindeutiger bestimmbar, als dies meist bei prozedurellem Prüfungscode möglich ist.

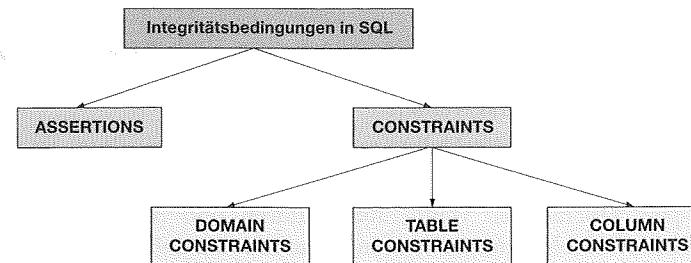


Abbildung 5.1: Integritätsbedingungen in SQL

Die SQL-Integritätsprüfung basiert auf zwei zentralen Konzepten, den ASSERTIONS und den CONSTRAINTS. ASSERTIONS sind eigenständige Datenbankobjekte, mit denen sich Einschränkungen an eine oder mehrere Tabellen formulieren lassen. CONSTRAINTS hingegen hängen unmittelbar von der Tabelle ab, für die sie definiert sind. Sie stellen bis auf die Fehlerkorrekturoption (Integrity Repair) der Fremdschlüssel syntaktische Kurzschreibweisen für ASSERTIONS dar. Je nach Art ihrer Definition werden die COLUMN CONSTRAINTS (Spaltenbedingung), die TABLE CONSTRAINTS (Tabellenbedingung) und die DOMAIN CONSTRAINTS (Domänenbedingung) differenziert. Die Unterscheidung zwischen diesen drei CONSTRAINT-Arten ist sowohl syntaktischer wie auch funktioneller Natur. Da in der Praxis meist nur die Tabellen- und Spaltenbedingungen umgesetzt sind, erläutern wir diese ausführlicher, während wir die Domänenbedingungen und ASSERTIONS anschließend nur kurz vorstellen.

5.3.2.1 Tabellen- und Spaltenbedingungen

Tabellenbedingungen (TABLE CONSTRAINTS) und Spaltenbedingungen (COLUMN CONSTRAINTS) werden noch weiter in verschiedene Typen unterteilt.

Tabelle 5.18

Integritätsbedingungen	
[NOT] NULL	Legt fest, dass eine Spalte (nicht) NULL, also leer sein kann, DEFAULT ist NULL.
UNIQUE	Bestimmt eine oder mehrere Spalten als eindeutigen Schlüssel. Die Werte dieser Schlüsselspalten erlauben keine Duplikate, dürfen jedoch NULL-Werte enthalten. Mehrere UNIQUE KEYS sind je Tabelle definierbar. Für die Schlüsselspalten wird automatisch ein INDEX (vgl. Kapitel 5.3.4 und 9) angelegt.
PRIMARY KEY	Bestimmt eine oder mehrere Spalten als Primärschlüssel im Sinne des relationalen Modells. Die Werte dieser Schlüsselspalten erlauben keine Duplikate und dürfen nicht NULL sein. Maximal ist ein Primärschlüssel je Tabelle definierbar. Für die Schlüsselspalten wird automatisch ein INDEX angelegt.
FOREIGN KEY	Bestimmt eine oder mehrere Spalten der Detailtabelle als Fremdschlüssel, die der referentielle Integrität genügen müssen. Sie realisieren 1:n-Beziehungen aus dem ER-Modell und sind die einzigen CONSTRAINTS mit Fehlerkorrektur. Als Spaltenbedingung wird mit der REFERENCES-Klausel der Bezug zur Fremdschlüsseltabelle hergestellt.
CHECK	Legt eine Bedingung für eine oder mehrere Spalten fest, die jeder Datensatz der Tabelle erfüllen muss. In SQL sind prädikatenlogische ¹⁷ Ausdrücke der 1. Ordnung mit SELECT-Anfragen auf andere Tabellen zulässig.

Es folgen die fehlenden Syntaxausdrücke für die CREATE-Table-Anweisung, die die Definition von CONSTRAINTS (Spaltenbedingungen, Tabellenbedingungen) beschreiben.

```

<Spaltenbedingung> ::= 
  [ CONSTRAINT Constraintname ] <Spaltenbedingungsausdruck>
  [ <CONSTRAINT Characteristika> ]

<Spaltenbedingungsausdruck> ::= NOT NULL
                                PRIMARY KEY
                                UNIQUE
                                <Referenzspezifikation>
                                CHECK ( <Suchbedingung> )

<Tabellenbedingung> ::= 
  [ CONSTRAINT Constraintname ] <Tabellenbedingungsausdruck>
  [ <CONSTRAINT Characteristika> ]

<Tabellenbedingungsausdruck> ::= PRIMARY KEY ( Spaltenname [ , Spaltenname ]... )
                                UNIQUE ( Spaltenname [ , Spaltenname ]... )
                                FOREIGN KEY ( Spaltenname [ , Spaltenname ]... )
                                <Referenzspezifikation>
                                | CHECK ( <Suchbedingung> )

<Referenzspezifikation> ::= REFERENCES Tabelle [ ( Spaltenname [ , Spaltenname ]... ) ]
                            <Übereinstimmungstyp>
                            [ <Fehlerkorrektur Definition> ]
  
```

¹⁷ Prädikatenlogische Ausdrücke erster Ordnung findet man z.B. in [Dassow 2005] erklärt.

Der Übereinstimmungstyp und die Fehlerkorrektur werden im Abschnitt 5.3.3 erklärt. Die für die CONSTRAINTS vergebenen Namen sollten sprechend sein, da sie in den Fehlermeldungen angezeigt werden. Die Option <CONSTRAINT Characteristika> wird in Abschnitt 5.3.3 unter dem Stichpunkt „Integritätsprüfung“ ausführlich erläutert, ebenso wie die Fremdschlüsseloptionen dort unter dem Stichpunkt „FOREIGN KEY-CONSTRAINTS und Fehlerkorrektur“.

Spaltenbedingungen gehören zu einer Spaltendefinition, während sich Tabellenbedingungen auf die gesamte Tabelle beziehen. Während in einer Spaltenbedingung nur ein einspaltiger PRIMARY KEY bzw. UNIQUE-CONSTRAINT definiert werden kann, können Tabellenbedingungen sich auch auf mehrere Spalten beziehen und damit auch mehrspaltige Schlüssel definieren. Die deutlich unterschiedliche Syntax der Fremdschlüsselbedingung, je nachdem, ob sie als Spaltenbedingung oder als Tabellenbedingung definiert ist, röhrt lediglich daher, dass bei Tabellenbedingungen explizit eine oder mehrere Fremdschlüsselspalten angegeben werden müssen, während sich die REFERENCES-Klausel der Spaltenbedingungen implizit auf die zugehörige Spalte bezieht.

Bei der <Suchbedingung> der CHECK-Bedingung handelt es sich um einen booleschen Ausdruck, wie er grundsätzlich auch in der WHERE-Bedingung von Anfragen formuliert werden kann (vgl. Abschnitt 5.5.5). Als Spaltenbedingung kann die Bedingung nur für die zugehörige Spalte definiert werden, als Tabellenbedingung für mehrere Spalten der Tabelle.

Da in den CHECK-Bedingungen (siehe Beispiel unten) für die Spalten der zugehörigen Tabelle keine Quantifizierung (als Allquantor oder Existenzquantor) spezifiziert wird, stellt sich die Frage nach der Semantik eines solchen Ausdrucks. Da Integrität für jeden Datensatz geprüft wird, ergibt sich eine implizite Allquantifizierung für die Spalten der zugehörigen Tabelle, d.h., die Checkbedingung muss für jeden Datensatz erfüllt sein. Für die Quantifizierung anderer Tabellen, auf die mittels SELECT zugegriffen wird, steht der EXISTS-Operator zur Verfügung, mit dem auch ein Allquantor simuliert werden kann (vgl. Abschnitt 5.5.9).

In der Praxis ist es vielfach eine Geschmacksfrage, außer der NOT NULL-Restriktion, alle Integritätsbedingungen übersichtlich am Ende der CREATE TABLE-Anweisung als Tabellenbedingung zu formulieren oder die Bedingungen möglichst unmittelbar bei der Spalte als Spaltenbedingung, bis auf die NOT-NULL-Restriktion, zu spezifizieren, die dadurch eingeschränkt wird. Allerdings kann man jede Spaltenbedingung auch als Tabellenbedingung aufschreiben, was umgekehrt nicht gilt, da sich Spaltenbedingungen nur auf eine Spalte beziehen können.

Beispiel

mit Spalten- und Tabellenbedingungen

```
CREATE TABLE Teile(
TNr          NUMERIC(38)  CONSTRAINT cc_Teile_pk PRIMARY KEY,
ME          VARCHAR(10),
Bezeichnung  VARCHAR(50) NOT NULL
                      CONSTRAINT cc_Teile_uk UNIQUE,
Typ          VARCHAR(50),
Herstellkosten  NUMERIC,
Einkaufspreis  NUMERIC,
Mindestbestand  NUMERIC,
Bestand      NUMERIC      CONSTRAINT cc_pruefe_bestand
                           CHECK (Bestand > 0),
Lieferzeit    NUMERIC,
```

```
Herstelldauer  NUMERIC,
Gewicht       NUMERIC,
Reserviert    NUMERIC,
Verfuegbar    NUMERIC,
Zeitstempel   DATE,
CONSTRAINT tc_bestaende CHECK (Bestand >= Mindestbestand),
CONSTRAINT tc_kosten_preis
           CHECK (Herstellkosten IS NOT NULL OR
                  Einkaufspreis IS NOT NULL),
CONSTRAINT tc_herstell_einkauf
           CHECK ((Herstellkosten IS NOT NULL AND
                  Herstelldauer IS NOT NULL)
                  OR (Einkaufspreis IS NOT NULL AND
                  Lieferzeit IS NOT NULL));
```

Alle als „cc_“ bezeichneten CONSTRAINTS sowie NOT NULL sind als Spaltenbedingungen definiert und die drei letzten CONSTRAINTS, die mit „tc_“ bezeichnet sind, als Tabellenbedingungen. Der CONSTRAINT cc_Teile_pk ist Primärschlüssel der Tabelle „Teile“ und bezieht sich nur auf die Spalte „TNr“. Die Spalte „Bezeichnung“ darf nicht leer sein und ist zudem ein Eindeutigkeitsschlüssel der Tabelle. Mit dem CONSTRAINT pruefe_bestand wird sichergestellt, dass für alle Datensätze der Bestand immer größer als 0 ist. Und mit tc_bestaende prüft man, ob für alle Datensätze der Bestand größer gleich dem Mindestbestand ist. Bei tc_kosten_preis wird sichergestellt, dass bei allen Datensätzen auf jeden Fall wenigstens ein Einkaufspreis oder die Herstellkosten angegeben werden. Es können aber auch in beiden Spalten Eingaben erfolgen. Eine etwas komplexere Bedingung stellt tc_herstell_einkauf dar, die dafür sorgt, dass die Herstellkosten und die Herstelldauer angegeben sind oder Einkaufspreis und Lieferzeit. Die Semantik überschneidet sich mit der von tc_kosten_preis, wobei es hier darum geht, unterschiedliche komplexe Bedingungen zu zeigen.

```
CREATE TABLE Struktur(
OTeil      NUMERIC(38)
           CONSTRAINT OTeil_fk REFERENCES Teile(TNr),
UTeil      NUMERIC(38),
Position   NUMERIC(38),
CONSTRAINT Struktur_pk PRIMARY KEY (UTeil, OTeil, Position),
Menge      NUMERIC(20,4) NOT NULL,
Ausschuss  NUMERIC(20,4),
Arbeitsgang  NUMERIC(38) NOT NULL,
Zeitstempel DATE,
CONSTRAINT UTeil_fk FOREIGN KEY (UTeil) REFERENCES Teile(TNr));
```

Die Spalten „Menge“ und „Arbeitsgang“ sind beides Pflichteingabespalten, also obligatorisch. Für die drei Primärschlüsselspalten „OTeil“, „UTeil“ und „Position“ muss keine explizite NOT NULL-Beschränkung definiert werden. Da es drei Spalten sind, wird der Primärschlüssel als Tabellenbedingung formuliert. Der Fremdschlüssel UTeil_fk hingegen könnte auch als Spaltenbedingung definiert werden, da er sich nur auf eine Spalte bezieht. Die beiden Fremdschlüsselbeziehungen OTeil_fk und UTeil_fk sichern die referentielle Integrität zu der Master-Tabelle „Teile“. Da die beiden Fremdschlüsselspalten OTeil und UTeil für die Detailtabelle als Primärschlüsselspalten definiert sind, entspricht diese Fremdschlüsselbeziehung einer identifizierenden 1:n-Beziehung im ER-Modell (vgl. Abschnitt 3.2.6 und folgende Abbildung).

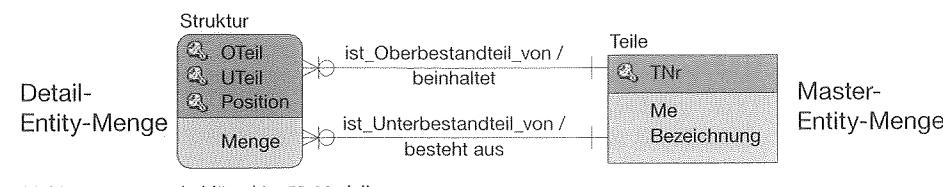


Abbildung 5.2: Fremdschlüssel im ER-Modell

5.3.2.2 Domänenbedingungen

Domänen (Wertebereiche, domains) geben zulässige Werte vor, die in der zugehörigen Spalte gespeichert werden können. Definiert werden sie als eigenständige Datenbankobjekte und können mit den Befehlen CREATE, ALTER und DROP angelegt, geändert oder gelöscht werden. Sie bestehen aus einem Datentyp, optional aus einem Default-Wert und optional aus einer oder mehreren Domänenbedingungen. Sie können anstelle von Datentypen für Spaltendefinitionen verwendet werden. Sie vereinfachen die Pflege umfangreicherer Datenmodelle, indem sie eine Möglichkeit darstellen, wiederkehrende Datendefinitionen zentral zu verwalten und dann z.B. bei CREATE TABLE beliebig oft wiederzuverwenden. Ein Beispiel für eine solche Domäne ist eine Domäne für monetäre Spalten mit dem Datentyp NUMBER(12,2), die für die Spalten Einkaufs- und Verkaufspreise, Rechnungssumme, Gehalt, Spesen etc. verwendet wird, und dem Default-Wert 0 sowie die Wertebereichseinschränkung „>=0“. Andere Beispiele sind Domänen für das Geschlecht mit „m“, „w“ oder den Wochen- oder Arbeitstagen von Samstag bzw. Montag bis Freitag. Als Domänenbedingung sind nur CHECK-Bedingungen zulässig, wobei der Platzhalter VALUE für den unbekannten Spaltennamen verwendet wird. Alle Domänenbedingungen können auch als Spaltenbedingung formuliert werden und ihre Prüfung funktioniert auch genauso.

Beispiel

für eine Domänenbedingung

```
CREATE DOMAIN geld_dom NUMERIC(12,2) DEFAULT 0 CHECK (VALUE >= 0);
```

5.3.2.3 ASSERTIONS

Eine ASSERTION ist ein eigenständiges Datenbankobjekt, das mit CREATE angelegt und mit DROP gelöscht werden kann. Eine ASSERTION formuliert eine CHECK-Bedingung. Allerdings hängt die Existenz der ASSERTION von der Existenz der Tabellen ab, auf die in der Bedingung Bezug genommen wird. Die ASSERTION existiert nur so lange, wie diese Tabellen vorhanden sind. Während bei den CHECK-Bedingungen das CONSTRAINT aufgrund der Syntax eine implizite Allquantifizierung für die Spalten der zugehörigen Tabelle vorgegeben ist, kann bei den ASSERTIONS darüber hinaus auch eine EXISTS-Quantifizierung formuliert werden. Nur bei den ASSERTIONS sind alle prädikatenlogischen Ausdrücke 1. Ordnung ohne Einschränkungen formulierbar.

Beispiel

für eine ASSERTION

Es soll die Integritätsbedingung „Alle Oberteile der Strukturtabelle sind Baugruppen oder Artikel“ formuliert werden, was mit simuliertem Allquantor (doppeltes NOT EXISTS¹⁸) heißt: „Es gibt keine OTeile in der Strukturtabelle, die nicht vom Typ Baugruppe oder Artikel sind.“

```
CREATE ASSERTION Oberteil_ass
  CHECK (NOT EXISTS (SELECT * FROM Struktur
    WHERE OTeil NOT IN (SELECT TNr FROM Teile
      WHERE Typ IN ('Baugruppe', 'Artikel'))));
```

Als Tabellen- oder Spaltenbedingung für die Spalte „OTeil“ der Tabelle „Struktur“ sieht die Anforderung (aufgrund der impliziten Allquantifizierung) so aus:

```
CHECK (OTeil IN (SELECT TNr FROM Teile
  WHERE Typ IN ('Baugruppe', 'Artikel')))
```

Anders sieht es bei der Anforderung aus: „Es muss wenigstens ein Teil geben, das kein Oberteil hat.“ Diese lässt sich aufgrund der impliziten Allquantifizierung in den Tabellen- und Spaltenbedingungen nicht formulieren, jedoch sehr wohl als ASSERTION.

```
CREATE ASSERTION Oberstes_Oberteil_ass
  CHECK (EXISTS (SELECT * FROM Struktur
    WHERE OTeil NOT IN (SELECT UTeil FROM Struktur)));
```

Nachfolgende Abbildung zeigt, welche aus der ER-Modellierung bekannten Integritätsbegriffe (Abschnitt 4.2.3) durch welche SQL-Konzepte realisiert werden:

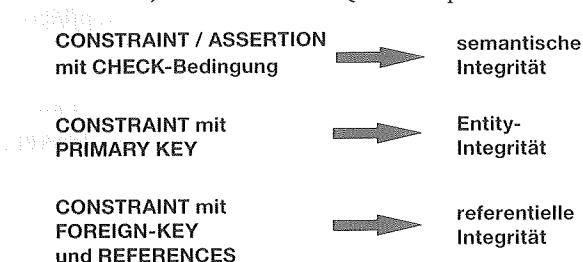


Abbildung 5.3: SQL-Integrität und Integrität des relationalen Modells

Domänen-, Spalten- und Tabellenbedingungen sowie die ASSERTIONS weisen hohe semantische Redundanzen untereinander auf. So stellen [NOT] NULL, PRIMARY KEY, UNIQUE, FOREIGN KEY Kurzschreibweisen für teilweise recht komplexe CHECK-Bedingungen dar. Einzige Ausnahme ist die Option zur Fehlerkorrektur bei den Fremdschlüsselbedingungen, die es für alle anderen Bedingungen nicht gibt. Des Weiteren lassen sich Domänenbedingungen in Spaltenbedingungen und diese in Tabellenbedingungen transformieren, die ihrerseits auch als ASSERTION ausgedrückt werden können. Es gibt jedoch ASSERTION-Bedingungen, die sich nicht in eine Tabellenbedingung überführen lassen.¹⁹

¹⁸ vgl. Abschnitt 5.5.9

¹⁹ Für eine ausführliche Diskussion der Semantik der verschiedenen SQL-Integritätsbedingungen verweisen wir auf [Behrend et al. 2001] und [Celko 2000].

5.3.3 Integritätsprüfung und Integritätsmonitor

Trotz aller syntaktischen Unterschiede läuft die Integritätsprüfung in SQL sowohl bei den CONSTRAINTS wie auch bei den ASSERTIONS nach dem gleichen Schema ab. Einige Ausnahme stellen die Fehlerkorrekturoptionen der Fremdschlüsselbedingungen dar, die anstelle des sonst üblichen Zurückrollens programmiert werden können. Drei wichtige Grundsätze gelten für alle Integritätsbedingungen:

- Eine Bedingung ist dann erfüllt, wenn sie nicht zu FALSE ausgewertet wird. Vor dem Hintergrund der dreiwertigen Logik heißt das, sie ist dann erfüllt, wenn sie zu TRUE oder NULL/UNKNOWN ausgewertet wird. Hier wird also ein unbekanntes Ergebnis wie TRUE behandelt. Anders sieht das bei der Anfrageauswertung aus, dort wird UNKNOWN wie FALSE interpretiert (vgl. Abschnitt 5.5.5).
- Wichtig für die grundsätzliche Konsistenz der Daten ist, dass beim Erzeugen von Integritätsbedingungen die Bedingung für die bereits vorhandene Datenbasis geprüft wird. Nur wenn alle gespeicherten Daten die Bedingung erfüllen, kann sie im Datenbanksystem erstellt werden, ansonsten wird sie abgewiesen.
- Die Integritätsprüfung für eine Anweisung wird immer, unabhängig davon, ob ein oder mehrere Datensätze geändert werden, „en block“ im Anschluss nach allen Datensatzänderungen durchgeführt. Hier bleibt der Grundsatz der Reihenfolgeunabhängigkeit strikt gewahrt (vgl. Abschnitt 7.3.6).

5.3.3.1 Der Integritätsmonitor

Das Datenbankmanagementsystem verfügt über eine zentrale Komponente zur Verwaltung und Prüfung der Integritätsbedingungen, den so genannten Integritätsmonitor.

Seine Aufgaben sind:

- die Verwaltung der in der Datenbank abgelegten Integritätsbedingungen,
- das Erkennen von prüfungsrelevanten Ereignissen (INSERT, UPDATE, DELETE),
- das Ermitteln der von diesen Ereignissen betroffenen Integritätsbedingungen,
- das Auswerten (Kontrollieren) der Bedingungen,
- die Interpretation der Ergebnisse der Integritätskontrolle sowie
- die Reaktion auf diese Ergebnisse (Zurückrollen, Fehlerkorrektur ...).

Das SQL-Integritätskonzept basierend auf den CONSTRAINTS und ASSERTIONS ist hundertprozentig sicher. Alle in einer Datenbasis gespeicherten Daten erfüllen immer die definierten Integritätsbedingungen. Und eben diese Sicherheit kann mit alternativen Ansätzen nicht gewährleistet werden. Eine Alternative ist die prozedurale Programmierung in Anwendungsprogrammen. Allerdings entsteht dabei das Problem, dass die Prüfung unter Umständen in verschiedenen Programmen erfolgen muss, mit der Folge von redundantem oder auch vergessenem Code. Eine andere Alternative ist die Einkapselung der Änderungsbefehle, wie man dies von den Methoden aus der objektorientierten Programmierung kennt. Für jede Änderungsanweisung auf jede Tabelle muss eine Extraprozedur geschrieben werden, die neben den Einfüge-/Änderungs-/Löschenbefehlen auch den für diese Operationen erforderlichen Prüfungscode beinhaltet. Es muss dann mittels der Rechteverwaltung gewährleistet werden, dass die Benutzer nur noch über diese Zwischenschicht Zugang zu den Daten haben. Interaktive Ad-hoc-Änderungen sind nicht mehr möglich. Ein Vorteil dieser beiden

alternativen Lösungen liegt jedoch in der flexiblen Programmierung der Reaktionen auf Integritätsfehler, die leider beim SQL-Standard unzureichend spezifiziert ist. Eine Diskussion der alternativen Lösung mittels Datenbanktrigger finden Sie im Abschnitt 7.3 bei den aktiven Datenbanksystemen.

5.3.3.2 SQL-Transaktion im Sinne der Integritätsprüfung

Ein zentraler Begriff für das Verständnis des SQL-Integritätskonzepts ist der der Transaktion. Eine Transaktion²⁰ ist eine Folge von Änderungsanweisungen, die als atomare Einheit ausgeführt wird. Im Sinne der Integritätsprüfung überführt sie eine konsistente Datenbasis in eine wiederum konsistente. Die atomare Ausführung einer Transaktion (ganz oder gar nicht) bedingt, dass sie mit zwei unterschiedlichen Befehlen beendet werden kann. Mit der COMMIT-Anweisung werden die Änderungen in der Datenbasis permanent gespeichert. Mit der ROLLBACK-Anweisung werden alle durch die Änderungsanweisungen verursachten Datenmanipulationen rückgängig gemacht. Der Zustand vor Transaktionsbeginn wird wiederhergestellt.

5.3.3.3 Integritätsprüfung

Für die Integritätsprüfung wird in SQL vorausgesetzt, dass zu Beginn einer Transaktion ein konsistenter DB-Zustand vorliegt. Dies ist eine äußerst wichtige Voraussetzung, damit die Konsistenz vom Integritätsmonitor effizient überwacht werden kann. Unter dieser Voraussetzung ist es ausreichend, dass sich der Monitor bei der Prüfung auf die während der Transaktion geänderten Daten beschränkt, statt alle Daten der Datenbasis zu kontrollieren. Aus dem SQL-Transaktionskonzept resultieren unmittelbar die beiden möglichen Prüfungszeitpunkte: IMMEDIATE und DEFERRED. Obwohl oben die <CONSTRAINT Characteristika> nur bei den Tabellen- und Spaltenbedingungen explizit im Syntaxdiagramm angegeben sind, sind sie auch für die Domänenbedingungen und die ASSERTIONS definierbar (vgl. Abschnitt 5.3.2).

```
<CONSTRAINT Characteristika> ::=  
/ INITIALLY DEFERRED | INITIALLY IMMEDIATE / [ NOT ] DEFERRABLE ]
```

Mit DEFERRED (verzögert) wird die Prüfung der Integritätsbedingung zum Ende der Transaktion (COMMIT) bezeichnet. IMMEDIATE (unmittelbar) beschreibt die Prüfung unmittelbar im Anschluss an die Ausführung einer einzelnen Änderungsanweisung. IMMEDIATE hat den Vorteil, dass Integritätsfehler so früh wie möglich erkannt werden, und DEFERRED, dass während der laufenden Transaktion durch nachfolgende Anweisungen ein zwischenzeitlich inkonsistenter Zustand wieder korrigiert werden kann. INITIALLY bezeichnet den Prüfungszeitpunkt ab dem Moment der Definition der Integritätsbedingung und [NOT] DEFERRABLE informiert darüber, ob eine ursprünglich IMMEDIATE definierte Bedingung während einer Transaktion mit der Anweisung SET CONSTRAINT auf den Zustand DEFERRED gesetzt werden kann. Default ist INITIALLY IMMEDIATE DEFERRABLE.

²⁰ Eine detaillierte Definition und eine Vorstellung der mit Transaktionen verbundenen Konzepte erfolgt in Kapitel 8.

```
<SET CONSTRAINTS Anweisung> ::=  
  SET CONSTRAINTS { ALL | Constraintname [ , Constraintname ]... }  
    { DEFERRED | IMMEDIATE };
```

Als Reaktion auf Integritätsfehler stellt für die Integritätsbedingungen PRIMARY KEY, UNIQUE, NOT NULL, CHECK das Zurückrollen die einzige mögliche Reaktion dar, für Fremdschlüssel besteht zusätzlich die Option der Fehlerkorrektur (s.u.).

5.3.3.4 Zurückrollen im Fehlerfall

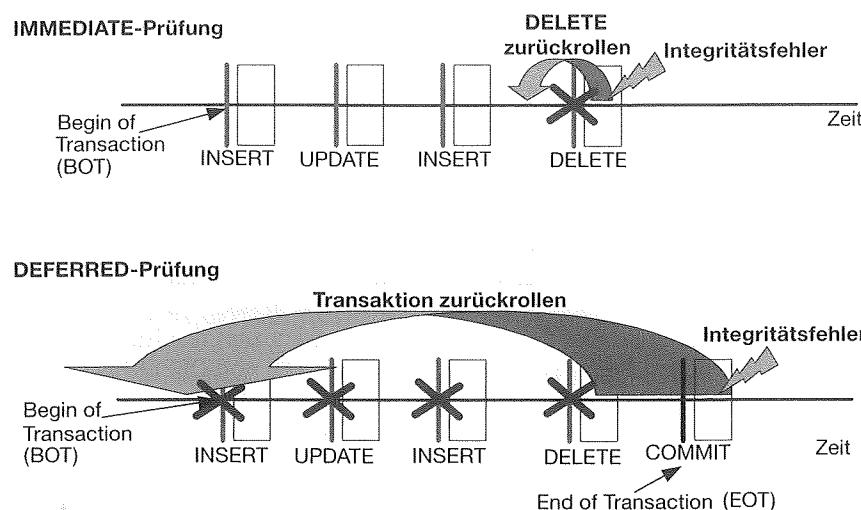


Abbildung 5.1: Zurückrollen bei Integritätsfehlern

Wie ▶ Abbildung 5.1 zeigt, wird bei einer IMMEDIATE-Prüfung unmittelbar nach jeder Datenänderung durch Änderungsanweisungen eine Integritätsprüfungsphase (weißes Kästchen) durchgeführt, während der alle betroffenen IMMEDIATE-Integritätsbedingungen ausgewertet werden. Wird ein Fehler festgestellt, dann wird die weitere Prüfung abgebrochen und die aktuelle fehlerverursachende Anweisung rückgängig gemacht. Dann liegt wieder die konsistente Datenbasis nach der letzten erfolgreichen Integritätsprüfung vor. Bei der Prüfung der DEFERRED-Integritätsbedingungen wird bis zur COMMIT-Anweisung gewartet, die die Transaktion beendet. Taucht dann während dieser abschließenden Prüfung bei den betroffenen verzögerten Integritätsbedingungen ein Fehler auf, dann wird die gesamte Transaktion rückgängig gemacht und es liegt die konsistente Datenbasis von vor Beginn der Transaktion wieder vor.

Beispiel

aus der Fahrrad-Welt Byce & Co.

```
CREATE TABLE Teile  
(TNr      NUMERIC(38),  
 Bezeichnung  VARCHAR(50),  
 ...);
```

```
CREATE TABLE Auftraege  
(AuftragsNr  NUMERIC(38),  
 Kun_Nr      NUMERIC(38),  
 ...);
```

```
CREATE TABLE Auftragspositionen  
(TNr      NUMERIC(38) NOT NULL  
CONSTRAINT FK_Teile  
REFERENCES Teile (TNr) ON DELETE SET NULL,  
AuftragsNr  NUMERIC(38) NOT NULL  
CONSTRAINT FK_Auftrag  
REFERENCES Auftraege (AuftragsNr) ON DELETE CASCADE,  
Menge      NUMERIC(38));
```

Auftragspositionen			Teile		Auftraege	
AuftragsNr	TNr	Menge	TNr	Bezeichnung	AuftragsNr	Kun_Nr
1	13	30	13	Klapprad	1	4711
2	14	200	14	Rennrad	2	4712
Detail			Master1		Master 2	

Wird der Auftrag mit der AuftragsNr 1 gelöscht, dann wird der Detaildatensatz mit der AuftragsNr 1 aus den Auftragspositionen automatisch gelöscht. Wird das Teil mit der Nummer 14 gelöscht, dann wird beim zugehörigen Detaildatensatz in den Auftragspositionen mit der TNr 14 diese Spalte geleert, d.h. auf NULL gesetzt. Ohne eine Fremdschlüsselpflicht gäbe es in beiden Fällen Detaildatensätze, deren Werte in den Fremdschlüsselspalten auf Werte in der Master-Tabelle verweisen, die nicht mehr existieren. Solche Detaildatensätze werden als „dangling tuples“ bezeichnet und stellen Probleme dar, die mit der Fehlerkorrektur aber gut behoben werden können.

5.3.3.7 Nachteile des SQL-Integritätskonzepts

Zu den Schwächen des SQL-Standards zählt neben der fehlenden Möglichkeit selbst programmierter Fehlerkorrektur insbesondere, dass es keine Möglichkeit gibt, transitorische oder gar temporale Integritätsbedingungen²¹ zu formulieren. Trotzdem bleibt die Integritätsprüfung insgesamt eine sehr nützliche Funktionalität, die eine konsistente Datenbasis für Entwickler auf komfortable Weise sicherstellt. Die Funktionalität hat natürlich ihren Preis und der ergibt sich aus der erhöhten Laufzeit für die Vielzahl an Prüfungen²², die durchgeführt werden müssen, und der daraus resultierenden höheren

²¹ vgl. Abschnitt 4.3

²² vgl. Kapitel 8.2

Beispiel

aus der Fahrrad-Welt Byce & Co.

```
CREATE TABLE Teile
(TNr      NUMERIC(38),
Bezeichnung  VARCHAR(50),
...);

CREATE TABLE Auftraege
(AuftragsNr  NUMERIC(38),
Kun_Nr      NUMERIC(38),
...);

CREATE TABLE Auftragspositionen
(TNr      NUMERIC(38) NOT NULL
CONSTRAINT Fk_Teile
REFERENCES Teile (TNr) ON DELETE SET NULL,
AuftragsNr  NUMERIC(38) NOT NULL
CONSTRAINT Fk_Auftrag
REFERENCES Auftraege (AuftragsNr) ON DELETE CASCADE,
Menge      NUMERIC(38));
```

Auftragspositionen			Teile		Auftraege	
AuftragsNr	TNr	Menge	TNr	Bezeichnung	AuftragsNr	Kun_Nr
1	13	30	13	Klapprad	1	4711
2	14	200	14	Rennrad	2	4712
Detail			Master 1		Master 2	

Wird der Auftrag mit der AuftragsNr 1 gelöscht, dann wird der Detaildatensatz mit der AuftragsNr 1 aus den Auftragspositionen automatisch gelöscht. Wird das Teil mit der Nummer 14 gelöscht, dann wird beim zugehörigen Detaildatensatz in den Auftragspositionen mit der TNr 14 diese Spalte geleert, d.h. auf NULL gesetzt. Ohne eine Fremdschlüsselprüfung gäbe es in beiden Fällen Detaildatensätze, deren Werte in den Fremdschlüsselspalten auf Werte in der Master-Tabelle verweisen, die nicht mehr existieren. Solche Detaildatensätze werden als „dangling tuples“ bezeichnet und stellen Probleme dar, die mit der Fehlerkorrektur aber gut behoben werden können.

5.3.3.7 Nachteile des SQL-Integritätskonzepts

Zu den Schwächen des SQL-Standards zählt neben der fehlenden Möglichkeit selbst programmierten Fehlerkorrekturen insbesondere, dass es keine Möglichkeit gibt, transitorische oder gar temporale Integritätsbedingungen²¹ zu formulieren. Trotzdem bleibt die Integritätsprüfung insgesamt eine sehr nützliche Funktionalität, die eine konsistente Datenbasis für Entwickler auf komfortable Weise sicherstellt. Die Funktionalität hat natürlich ihren Preis und der ergibt sich aus der erhöhten Laufzeit für die Vielzahl an Prüfungen²², die durchgeführt werden müssen, und der daraus resultierenden höheren

21 vgl. Abschnitt 4.3

22 vgl. Kapitel 8.2

Belastung des Datenbank-Servers. Es obliegt dem Entwickler, im Einzelfall zu beurteilen, ob es Probleme gibt und wo für die jeweilige Anwendung die Prioritäten liegen.

5.3.3.8 Integritätskonzept bei Oracle

Das SQL-Integritätskonzept ist nur rudimentär umgesetzt. ASSERTIONS und Domänenbedingungen fehlen gänzlich. Die CHECK-CONSTRAINTS, definiert als Tabellen- oder als Spaltenbedingung, können nur Spalten der zugehörigen Tabelle oder einer konstanten Wertemenge miteinander vergleichen. SELECT-Anfragen sind nicht zulässig, was eine sehr starke Beschränkung der Ausdrucks Kraft darstellt. Fremdschlüsselbedingungen verfügen auch nur über eine beschränkte Fehlerkorrektur. Die ON UPDATE-Klausel fehlt gänzlich und die ON DELETE-Klausel verfügt nur über die beiden Optionen CASCADE und SET NULL. Primär- und Eindeutigkeitsschlüssel sind wie beim Standard vorgegeben implementiert. Die Spaltenbedingungen heißen „inline_constraint“ und die Tabellenbedingungen „out_of_line_constraint“. Die CONSTRAINTS sollten mit einem Namen bezeichnet werden, sonst vergibt das DBMS einen systemeigenen Namen der Gestalt SYS_Cn, wobei n eine fortlaufende Nummer ist.²³ Alle Informationen über die Integritätsbedingungen werden in den DICTIONARY-Tabellen USER_CONSTRAINTS, USER_CONS_COLUMNS und USER_CONS_OBJ_COLUMNS abgespeichert.

Oracle kennt noch ein zusätzliches De-/Aktivierungskonzept für Integritätsbedingungen. Bei den CREATE und ALTER TABLE-Befehlen steht eine DISABLE-/ENABLE-Option zur Verfügung (vgl. Abschnitt 5.3.10). Braucht man für bestimmte Aktionen einige der CONSTRAINTS nicht, so muss man sie nicht gleich aus dem Datenbanksystem löschen, man kann sie vorübergehend deaktivieren. Bei der anschließenden Aktivierung wird genau wie beim Erzeugen die Integritätsbedingung für alle relevanten Daten geprüft und nur dann aktiviert, wenn alle Datensätze die Bedingung erfüllen²⁴.

Bei der eigentlichen Prüfung gibt es auch einen feinen Unterschied: Es werden die von einer Änderungsanweisung betroffenen unmittelbaren Integritätsbedingungen nicht „en block“ nach allen Datensatzänderungen ausgeführt, sondern immer abwechselnd erst ein Datensatz geändert, dann die durch ihn betroffenen unmittelbaren Bedingungen geprüft und dann der nächste Satz geändert und die von ihm betroffenen unmittelbaren Bedingungen geprüft. Diese verschachtelte Ausführung verletzt die in SQL strikt gewährte Reihenfolgeunabhängigkeit (vgl. Abschnitt 7.3.6).

5.3.3.9 Integritätskonzept bei MySQL

Traditionell verfolgt MySQL ein völlig anderes Integritätskonzept als der SQL-Standard. Von Anfang an wurde eine automatische vom System vorgegebene Fehlerkorrektur betrieben, statt einer Fehlerentdeckung wie bei SQL. Erst seit kurzem mit der Version 5.0.2 wird zusätzlich ein zu SQL analoges Integritätskonzept angeboten.

5.3.3.10 Automatische Fehlerkorrektur als Default-Verhalten

Diese ist motiviert durch das Anwendungsszenario, dass in einer nicht transaktions-sicheren Tabelle (kein ROLLBACK, vgl. Abschnitt 8.6) eine Vielzahl an Einfügungen oder Änderungen durchgeführt werden sollen. Was ist zu tun, wenn ein Fehler auftritt?

23 Weitere Detailinformationen finden Sie in [ORACLE SQL 2005, Kap. 8].

24 Diese Zustände können dann noch mit den Optionen NO-/VALIDATE kombiniert werden, so dass es u.a. möglich wird, dass beim Aktivieren nicht kontrolliert wird. Vertiefende Informationen finden Sie bei [ORACLE SQL 2005, Kap. 8].

Zurückrollen ist nur schwer möglich, da die Änderungen ja teilweise persistent gespeichert sind. Durchgeführte Änderungen in der Datenbank belassen und den Rest der Manipulationen abbrechen? Damit ist bei Massenmanipulationen auch niemandem geholfen. Hier wird eindeutig die Prämisse gesetzt, dass eine Massenmanipulation der Daten nicht durch einige wenige Fehler aufs Spiel gesetzt werden darf – also Fehler möglichst korrigieren und dann weiterarbeiten. Hier ein Auszug aus der vom System vorgegebenen automatischen Korrekturstrategie der „best possible values“²⁵: Eine Möglichkeit für den Benutzer, selbst definierte Fehlerkorrekturen vorzugeben, besteht nicht.

Bei Bereichsüberschreitungen von

- numerischen Spalten wird anstelle des fehlerhaften Werts der kleinst- bzw. größtmögliche Wert gespeichert.
- alphanumerischen Spalten wird NULL bzw. soviel wie möglich von der zu großen Zeichenkette gespeichert.
- Datumsspalten werden fehlerhafte Datumswerte (z.B. 32.12.) ohne Korrektur gespeichert. Nur bei Werten, die nicht als Datum darstellbar sind, wird ein NULL-Datum gespeichert.

Fehlen Werte für

- NULL-Spalten, so wird der explizite Default-Wert genommen, falls er spezifiziert ist, sonst der implizite (0 bei numerischen und die leere Zeichenkette bei String-Datentypen). (Dieses Verhalten gilt auch für SQL und Oracle, wobei der implizite Default-Wert immer NULL ist.)
- NOT NULL-Spalten und wurde eine einzeilige INSERT-Anweisung verwendet, so wird die Anweisung fehlerhaft abgebrochen, bei mehrzeiligen INSERTS wird der implizite Default-Wert gespeichert.

Wird in einer

- numerischen Spalte eine Zeichenkette eingefügt, die nicht mit einer Zahl beginnt, dann wird 0 gespeichert, sonst die Zahl, mit der die Zeichenkette beginnt.
- ENUM-Spalte ein nicht passender Wert eingefügt, so wird der entsprechende Default-Wert genommen (vgl. Abschnitt 6.1.10).
- SET-Spalte ein nicht passender Wert eingefügt, so wird der fehlerhafte Wert ignoriert.

Durch eine Modusänderung kann ab der Version 5.0.2 aber auf eine SQL-konforme Integritätsprüfung umgeschaltet werden. Da wir dieses Thema nicht weiter vertiefen können, hier nur einige Modi, die recht umfassend die Funktionalität ändern²⁶:

- **ANSI:** Syntax und Semantik orientieren sich stärker an der von SQL.
- **TRADITIONAL:** Ein Verhalten wie „traditionelle“ Datenbanksysteme, was bei der Integritätsprüfung z.B. heißt, dass Fehlermeldungen und sofortiger Abbruch der Änderungsanweisung statt Warnungen durchgeführt werden.
- **STRICT_TRANS_TABLES:** Für Tabellen transaktionssicherer Speichermaschinen wie InnoDB wird bei einem Integritätsfehler die Anweisung abgebrochen und die gesamte Anweisung zurückgerollt. Für Tabellen nicht transaktionssicherer Speichermaschinen (zurzeit alle anderen) bricht die Anweisung nur ab, wenn der Feh-

²⁵ vgl. [MySQL 2006, Kap. 1.9.6.2]

²⁶ Andere Modi sind teilweise sehr begrenzt, was die an- oder ausgeschaltete Funktionalität angeht [MySQL 2006, Kap. 5.2.6].

ler in der ersten Datenänderung auftritt. Bei späteren Datenänderungen wird korrigiert und weitergearbeitet, da die bereits vorgenommenen Änderungen nicht mehr rückgängig zu machen sind.

- **STRICT_ALL_TABLES:** Modifiziert den Modus STRICT_TRANS_TABLES derart, dass auch bei Tabellen nicht transaktionssicherer Speichermaschinen die Anweisung immer abbricht, auch wenn bereits Änderungen gespeichert wurden. Dies hat zur Folge, dass Anweisungen mit Integritätsfehlern nur teilweise ausgeführt werden. Die nur teilweise Ausführung von Änderungsanweisungen widerspricht dem Paradigma der Reihenfolgeunabhängigkeit im SQL-Standard.

Umgesetzt wird der Modus global bzw. für eine Sitzung mit der SET SQL_MODE-Anweisung, die zahlreich auch bei den gespeicherten Routinen im Abschnitt 7.2 zum Einsatz kommt. Als Schlüsselwort für <Modusname> kommt u.a. einer der oben genannten Modi in Frage. Wir haben hier für Übungen und Beispiele mit dem Modus „TRADITIONAL“ gearbeitet.

```
<SET SQL_MODE Anweisung> ::=  
SET [ GLOBAL | SESSION ] SQL_MODE='<Modusname> ';
```

5.3.3.11 SQL-konforme Integritätsprüfung

MySQL hat die SQL-CONSTRAINTS [NOT] NULL, PRIMARY KEY, UNIQUE, FOREIGN KEY als Tabellen- und Spaltenbedingungen implementiert, Domänenbedingungen und ASSERTIONS sind jedoch auch nicht realisiert. Es sind bei den Spaltenbedingungen lediglich die Reihenfolge der DEFAULT-Option und des [NOT] NULL gegenüber SQL vertauscht und optional können nicht SQL-konforme Schlüsseldefinitionen verwendet werden: UNIQUE [KEY] statt UNIQUE und [PRIMARY] KEY statt PRIMARY KEY. Bei den Tabellenbedingungen ist es nur die UNIQUE [KEY]-Syntax, die differieren kann. In diesem Zusammenhang werden die beiden Begriffe KEY und INDEX völlig synonym verwendet. Bis auf diese syntaktischen Abweichungen, sind [NOT] NULL, Primär- und Eindeutigkeitsschlüssel SQL-konform implementiert. Für CHECK-Bedingungen gibt es aus Gründen der Kompatibilität mit anderen Datenbanksystemen zwar eine entsprechende Syntax, die programmiert werden kann, aber alle Speichermaschinen ignorieren sie bei der Prüfung. Dieser Mangel kann durch ENUM und SET-Datentypen etwas ausgeglichen werden, die es zumindest erlauben, Wertebereichsmengen vorzugeben (vgl. Abschnitt 6.1.10). Damit sich diese Datentypen aber auch wie richtige Integritätsbedingungen verhalten, muss wie zuvor bereits ausgeführt der SQL-Modus entsprechend gesetzt sein. Der ENUM-Datentyp würde dann dem Oracle-CHECK mit einem konstanten Wertebereich – ohne Vergleich mit anderen Spalten – entsprechen.

Für eine Fremdschlüsselbedingung müssen beide beteiligten Tabellen mit der Speichermaschine (Storage Engine vgl. Abschnitt 1.3 und 8.6)²⁷ InnoDB verwaltet werden. Die Syntax als Tabellen- und Spaltenbedingung ist insoweit SQL-konform, als nur die SQL-Option SET DEFAULT für die Fehlerkorrektur fehlt. Per Hand müssen für die Fremdschlüsselspalten beim Detail wie auch bei den zugehörigen Spalten der Master-Tabelle Indizes angelegt werden. Unter anderem für das Laden von Massendaten besteht die Möglichkeit, die Fremdschlüsselprüfung an- und auszuschalten. Dies gilt

²⁷ Für Release 5.2 sind die FOREIGN KEYS auch für die Storage Engine MyISAM angekündigt.

für alle Fremdschlüssel und nicht für einzelne. Die Informationen über angelegte Integritätsbedingungen werden im INFORMATION SCHEMA, das in jeder Installation enthalten ist, im MySQL-Dictionary²⁸ in den Tabellen TABLE_CONSTRAINTS und KEY_COLUMN_USAGE hinterlegt.

```
<SET FOREIGN_KEY_CHECKS Anweisung> ::=  
  SET FOREIGN_KEY_CHECKS = { 0 | 1 };
```

Auch MySQL kennt ein (De-)Aktivierungskonzept für Integritätsbedingungen, es ist nur etwas grober als bei Oracle. Beim ALTER TABLE-Befehl besteht die Möglichkeit für eine Tabelle, mit der Option ENABLE KEYS alle Schlüssel der Tabelle zu aktivieren bzw. mit DISABLE KEYS alle zu deaktivieren.

5.3.3.12 Prüfungsablauf

Der Ablauf der Prüfung hängt bei MySQL von verschiedenen Faktoren ab.

- MySQL unterstützt keine unterschiedlichen Integritätsprüfungszeitpunkte. Die Option für die <constraint characteristics> IMMEDIATE und DEFERRED fehlt, mit der Konsequenz, dass es keine DEFERRED-Prüfung zum COMMIT-Zeitpunkt gibt.
- Sind aufgrund der Ausführung von mehrzeiligen Manipulationsanweisungen Integritätsbedingungen zu prüfen, dann werden diese Bedingungen unmittelbar nach jedem Datensatz geprüft und nicht wie bei SQL „en block“ je Anweisung einmal. Dies stellt eine Verletzung der Reihenfolgeunabhängigkeit dar, die für einige besondere Eigenschaften bei der Prüfung und Korrektur von Fremdschlüsseln in Kauf genommen wird.
- Wird als Speichermaschine die InnoDB mit ihrem Transaktionskonzept verwendet, dann wird bei jedem Integritätsfehler aufgrund der zwingenden IMMEDIATE-Prüfung immer nur die fehlerhafte Datenmanipulation zurückgerollt.
- Bei allen anderen Speichermaschinen (alle ohne Transaktionskonzept) wird je nach gewähltem Modus die Ausführung des aktuellen Befehls beim fehlerhaften Datensatz abgebrochen. Alle bereits durchgeföhrten fehlerfreien Änderungen bleiben jedoch in der Datenbasis gespeichert, da sie ja bereits persistent gespeichert wurden. Dieses Verhalten der teilweisen Ausführung von Änderungsanweisungen ist nicht SQL-konform und nur möglich, da direkt nach jeder Datensatzänderung die Integrität kontrolliert wird. Auch dieses Verhalten stellt eine Verletzung der Reihenfolgeunabhängigkeit dar.
- Will man trotz Integritätsfehler auch die restlichen Manipulationen durchführen, so kann man bei den INSERT- und UPDATE-Anweisungen eine IGNORE-Option setzen. Dann wird trotz Fehler weitergearbeitet, nur die fehlerhaften Datensätze sind dann nicht in der Datenbasis zu finden.

²⁸ vgl. Abschnitt 5.8.2

5.3.3.13 Nachteile kommerzieller Datenbanksysteme (Oracle, MySQL)

Über den Standard hinausgehende Funktionalität wurde nicht realisiert, so dass die Schwächen des SQL-Standards auch die Schwächen der kommerziellen Datenbanksysteme sind. Zudem fehlt bei beiden betrachteten Systemen die Möglichkeit, ASSERTIONS zu definieren sowie in CHECK-Bedingungen mit SELECT-Anfragen auf andere Tabellen zuzugreifen. Bei MySQL sind CHECK-Bedingungen noch gar nicht prüfbar und dort fehlt auch eine Prüfung zum Transaktionsende. Zudem sind bei beiden Systemen die SQL-Vorgaben für die Fehlerkorrektur nicht vollständig umgesetzt. Diese Mängel stellen gravierende Einschränkungen der Ausdrucksstärke gegenüber den Möglichkeiten im SQL-Standard dar. Diese Schwächen lassen sich teilweise mittels aktiver Regeln ausgleichen (vgl. Abschnitt 7.3).

5.3.4 Die CREATE INDEX-Anweisung

Ein Index ist ein Datenbankobjekt, mit dem der lesende Zugriff auf Tabellen beschleunigt werden kann. Indizes stellen Implementierungen der Schlüsselbegriffe aus dem relationalen Modell²⁹ dar: Primär- und (eindeutige) Zweitenschlüssel. Für die Integritätsbedingungen PRIMARY KEY und UNIQUE wird automatisch ein eindeutiger Index angelegt, für Fremdschlüssel jedoch nicht (vgl. Abschnitt 5.3.2). Wie Indizes aufgebaut sind, wird im Rahmen der ISAM-Speicherstruktur im Abschnitt 9.3 ausführlich erläutert. Da der Index einer Tabelle eine separate Speicherstruktur ist, ergeben sich zwei Hauptvorteile:

- 1 Die Datensätze in einer Tabelle können lediglich nach einem Kriterium sortiert werden. In einem Index können die Spaltenwerte nach zusätzlichen Kriterien sortiert sein, was die Suche aufgrund optimierter Strategien ungeheuer beschleunigen kann.
- 2 Das Lesen von der Festplatte ist eine langwierige Operation. Liest man im Rahmen einer Suche nun die Datensätze selbst, so kann man aufgrund der verhältnismäßig großen Datenlänge in einem physikalischen Block nur wenige Suchinformationen mit einer Leseoperation in den Hauptspeicher holen. Da ein Indexeintrag meist nur wenige und nur relativ kurze Indexwerte und die kurze Adresse umfasst, können mit einer einzelnen Leseoperation unvergleichlich mehr Suchinformationen geholt werden.

```
<CREATE INDEX Anweisung> ::=  
  CREATE [UNIQUE] INDEX Indexname ON Tabellenname  
    (Spaltenname [, Spaltenname]...);
```

UNIQUE bewirkt, dass die Werte in der entsprechenden INDEX-Spalte eindeutig sind.

²⁹ vgl. Abschnitt 4.2.2

Beispiele

`CREATE UNIQUE INDEX Teile_il ON Teile (TNr);`
definiert einen eindeutigen Index auf der Teiletabelle.

`CREATE INDEX Struktur_il ON Struktur (OTeil, UTeil);`
definiert einen mehrspaltigen Index über die Spalten OTeil und UTeil der Tabelle „Struktur“. Beide Indizes entsprechen einem Zweitschlüssel im relationalen Modell.

Es ist nicht sinnvoll, auf zu vielen Spalten einer Tabelle Indizes anzulegen, da sie zwar den lesenden Zugriff beschleunigen, aber den schreibenden Zugriff abbremsen und natürlich zusätzlichen Speicherplatz verbrauchen.

Richtlinien zum Anlegen eines Index: Wann sollte man einen Index anlegen?

- Die Spalte wird häufig als Suchbedingung (WHERE-Klausel) oder zum Sortieren in ORDER-BY-Klauseln verwendet.³⁰
- Die Spalte wird häufig als JOIN-Bedingung zur Verbindung unterschiedlicher Tabellen (Fremdschlüsselspalten) benutzt.
- Die Spalte enthält einen großen Bereich an unterschiedlichen Werten und nur wenige NULL-Werte.
- Die Tabelle hat viele Datensätze (Richtwert: 100 Kbyte).
- Die Daten der Tabelle werden häufiger gelesen als verändert.

Die Verwendung von Indizes zur Optimierung von Anfragen wird etwas ausführlicher im Kapitel über SQL-Tuning³¹ behandelt.

5.3.5 Die Anweisungen CREATE VIEW und CREATE TABLE AS SELECT

VIEWS sind Benutzersichten definiert auf Tabellen oder auch auf anderen Sichten, die als Abfragen in der Datenbank gespeichert werden. Sie dienen dazu, Daten aus einer oder mehreren Tabellen/Sichten für bestimmte Zwecke, z.B. für ein Formular, zusammenzustellen. Sie sind insbesondere ein Arbeitsmittel der externen Ebene des ANSI-3-Ebenen-Modells (vgl. Abschnitt 1.8.2). Ebenfalls sehr häufig werden sie für die problemspezifische Aufbereitung der Daten für Datawarehouse-Applikationen (DWH), Datenreplikationen oder andere Informationssysteme verwendet. Auf Sichten können wie auf Tabellen auch Benutzerrechte vergeben werden. Sie stellen somit eine Möglichkeit dar, für bestimmte Anwender nur einen eingeschränkten Zugriff auf die Daten zuzulassen. Die Anweisung CREATE TABLE AS SELECT hat eine ähnliche Funktion wie die Anweisung CREATE VIEW AS SELECT, mit dem Unterschied, dass die abgeleitete Tabelle nicht nur als SELECT-Anweisung, sondern wie eine normale Tabelle gespeichert wird. Diese Anweisung gab es unter Oracle und auch unter MySQL schon lange, bevor sie in den aktuellen SQL2003-Standard aufgenommen wurde.

Nach Art der Datenhaltung werden verschiedene Arten von Sichten und generierte Tabellen unterschieden.

³⁰ vgl. Abschnitt 5.5.5 und 5.5.8

³¹ vgl. Abschnitt 5.5.12

5.3.5.1 Virtuelle Sichten (VN)

Für die virtuelle Sicht wird nur der Anfrageausdruck der Sichtdefinition im Data Dictionary gespeichert, nicht aber die Daten. Für das nachfolgende Beispiel der Sicht „Angestellten_kopie“ heißt das, dass die Angestelltendatensätze nur in der Tabelle „Angestellte“ vorliegen. Die Daten werden bei jedem lesenden Zugriff auf die Sicht gemäß dem Anfrageausdruck neu ermittelt. Es ist hier ein beliebiger, gültiger SQL-Anfrageausdruck zulässig (vgl. Abschnitt 5.5.11). Dieses Konzept ist im SQL-Standard definiert und bei Oracle und MySQL (erst seit Version 5.1) entsprechend realisiert.

Vorteil: Die Daten werden nicht redundant gespeichert.

Nachteil: Da bei einer Anfrage an die Sicht die Daten der Sicht erst ermittelt werden müssen, können bei komplexen SELECT-Ausdrücken bzw. bei großen Datenmengen sehr lange Laufzeiten entstehen.

5.3.5.2 Materialisierte Sichten (MS)

Beim Erstellen einer materialisierten Sicht wird nicht nur ihr Anfrageausdruck im Dictionary gespeichert, sondern auch alle Datensätze, die gemäß der Unteranfrage ermittelt werden. Bei Oracle ist ein solches Konzept mit teilweise eingeschränkten SELECT-Anfragen bereits realisiert, in MySQL noch nicht. Vorgaben vom SQL-Standard gibt es auch noch keine.³²

Vorteil: Da die Daten der Sichten wie bei den Tabellen gespeichert sind, dauert ein Sichtzugriff nicht länger als ein Tabellenzugriff.

Nachteil: Aufgrund der redundanten Datenspeicherung in Tabellen und Sichten treten Probleme der Aktualität und Konsistenz der Daten auf. Werden die zugrunde liegenden Tabellendaten aktualisiert, müssen auch die abgeleiteten Sichtdaten effizient aktualisiert werden. Hier gibt es bei Oracle bereits Ansätze, wie die Aktualisierung automatisch ausgeführt werden kann (Änderungspropagierung).

5.3.5.3 Generierte Tabellen (GT)

Es wird eine Kopie einer oder mehrerer vorhandener Tabellen mit den gewünschten Spaltendefinitionen und Daten angelegt. Die Daten der ursprünglichen Tabellen und der neu generierten Tabelle sind vollständig unabhängig voneinander und daher auch redundant gespeichert. Generierte Tabellen waren schon lange implementiert (Oracle, MySQL und andere), bevor sie erst 2003 in den SQL-Standard aufgenommen wurden.

Vorteil: Da keine Sicht, sondern eine physische Tabelle erzeugt wird, ist ein schneller Zugriff möglich.

Nachteil: Aufgrund der redundanten Datenspeicherung treten Probleme der Aktualität und Konsistenz der Daten auf. Für generierte Tabellen gibt es keine Ansätze einer automatischen Aktualisierung (Änderungspropagierung) wie bei den materialisierten Sichten.

³² Im Standard SQL2003 ist ein MATERIALIZED VIEW-Konzept für die nächste Version angekündigt. Die fehlende Standardisierung hat zu sehr unterschiedlichen Konzepten geführt. Bei der DB2 von IBM werden sie als SUMMERIZED TABLE bezeichnet.

Während es bei der Änderungspropagierung um die Aktualisierung der Sichtdaten geht, wenn sich die Tabellendaten geändert haben, geht es bei dem View-Update-Problem³³ um den umgekehrten Fall. Grundsätzlich können auch auf Sichten INSERT-, UPDATE- und DELETE-Anweisungen ausgeführt werden. Nur in sehr einfachen Fällen (eine Mastertabelle und ohne Aggregation) kann das Datenbankmanagementsystem diese geänderten Sichtdaten auf die zugrunde liegenden Tabellen automatisch und semantisch korrekt verteilen. Bei komplexeren Sichten (mehrere Master-Tabellen, Aggregation, Duplikate ...) kann dies nur „manuell“ vom Anwender mit seinem Hintergrundwissen (Wissen über die Semantik der Anwendung) durchgeführt werden. Ein wichtiges Hilfsmittel für einen Workaround sind die INSTEAD-OF-Datenbanktriggger, die es aber nur bei Oracle gibt (vgl. Abschnitt 7.3).

```
<CREATE VIEW Anweisung> ::=  
CREATE VIEW Sichtname [ ( Spaltenname [, Spaltenname ]... ) ]  
AS <Anfrageausdruck>;  
  
<CREATE TABLE AS SELECT Anweisung> ::=  
CREATE TABLE Tabellename [ ( Spaltenname [, Spaltenname ]... ) ]  
AS <Anfrageausdruck> [WITH [NO] DATA];
```

Beispiele

```
CREATE VIEW Angestellte_kopie AS SELECT * FROM Angestellte;  
  
CREATE VIEW Angestellte_adresse AS  
SELECT Nachname, Vorname, Strasse, Ort FROM Angestellte;  
  
CREATE VIEW Angestellte_Abteilung_Sicht (Mitarbeitername, Abteilung) AS  
SELECT Nachname || ' ' || Vorname, Abteilungen.Name  
FROM Angestellte, Abteilungen  
WHERE Angestellte.Abt_nr = Abteilungen.Abt_nr;  
  
CREATE TABLE Angesteller_GT AS SELECT * FROM Angestellte;
```

Die erste Anweisung generiert eine virtuelle Sicht als Kopie der Angestelltentabelle. Die zweite und die dritte Anweisung erstellen jeweils eine Sicht, die nur bestimmte Informationen der Angestellten anzeigen. Somit wird – bei entsprechender Rechtevergabe – vermieden, dass bestimmte Anwender z.B. so sensible Daten wie das Gehalt eines Mitarbeiters sehen können.³⁴ Die letzte Anweisung generiert eine Tabelle aus einer vorhandenen Tabelle, ohne eine Datenverbindung zwischen den Tabellen zu erzeugen.

Für die Spalten der Sichten und generierten Tabellen können neue Namen vergeben werden. Ihre Definition ergibt sich aus den Spaltendefinitionen der zugrunde liegenden Tabellen. Als Anfrage ist, wie auch bei den materialisierten Sichten, die vollständige SELECT-Syntax mit WITH-Klausel zulässig (vgl. Abschnitt 5.5.11). Die Option [WITH [NO] DATA] bewirkt, dass die generierten Tabellen mit bzw. ohne Daten angelegt werden, mit Daten ist die DEFAULT-Einstellung.

³³ vgl. Abschnitt 1.1.5

³⁴ Einzelheiten zur SELECT-Anweisung finden Sie im Abschnitt 5.5.

```
<Oracle->CREATE MATERIALIZED VIEW Anweisung> ::=  
CREATE MATERIALIZED VIEW Sichtname [ ( Spaltenname [, Spaltenname ]... ) ]  
[ <Aktualisierungsklausel> ] AS <Anfrageausdruck>;  
  
<Aktualisierungsklausel> ::=  
NEVER REFRESH  
| / REFRESH [ FAST | COMPLETE | FORCE ]  
[ ON DEMAND | ON COMMIT | / [ START WITH Datum ] NEXT Datum ] ]
```

Gegenüber der Definition einer virtuellen Sicht verfügt die Definition einer materialisierten Sicht zusätzlich über eine Aktualisierungsklausel. Mit der Aktualisierungsklausel werden der Zeitpunkt der Aktualisierung der materialisierten Daten bei Änderung der Daten in den zugrunde liegenden Tabellen wie auch die vier verschiedenen Arten spezifiziert:

- NEVER REFRESH: Es wird nie eine Aktualisierung der materialisierten Daten durchgeführt (analoges Verhalten wie bei GTs).
- COMPLETE: Der Anfrageausdruck wird vollständig neu ausgewertet und alle Datensätze werden unabhängig von den Änderungen in den Tabellen erneut ermittelt (vollständige Rematerialisierung).
- FAST: In Abhängigkeit von den Änderungen in den zugrunde liegenden Tabellen, die in speziellen Log-Dateien protokolliert werden, werden nur die von diesen Änderungen betroffenen materialisierten Datensätze aktualisiert (inkrementelle Aktualisierung). Die FAST-Option ist bei Oracle nur bei sehr speziellen materialisierten Sichten anwendbar.
- FORCE: Ist ein FAST-Refresh möglich, so wird er durchgeführt, sonst ein COMPLETE-Refresh. Das DBMS trifft diese Entscheidung.

Als Aktualisierungszeitpunkte bietet Oracle an:

- ON COMMIT: Die Aktualisierung wird automatisch zum Transaktionsende (COMMIT-Anweisung) durchgeführt.
- ON DEMAND: Die Aktualisierung wird automatisch durchgeführt, wenn spezielle Prozeduren des Package DBMS_MVIEW zur Aktualisierung von Sichten aufgerufen werden.
- NEXT DATE: Es wird ein Intervall für die Aktualisierung spezifiziert.
- Mit der START WITH-Option kann ein Starttermin für die Aktualisierungen definiert werden.

Alle weiteren Optionen des CREATE MATERIALIZED VIEW-Befehls sind in der Oracle-Online-Hilfe³⁵ zu finden.

Beispiel

einer materialisierten Sicht (Oracle)

```
CREATE MATERIALIZED VIEW Angestellte_Abteilung_mv
  (Mitarbeitername, Abteilung)
REFRESH FAST ON COMMIT AS
SELECT Nachname || ' ' || Vorname, Abteilungen.Name
FROM Angestellte, Abteilungen
WHERE Angestellte.Abt_nr = Abteilungen.Abt_nr
  AND Gehalt > 8000;
```

5.3.6 Die CREATE SEQUENCE-Anweisung

Eine Sequenz ist ein Datenbankobjekt, mit dem eindeutige, fortlaufende Nummern erzeugt werden können, die in der Regel für Schlüsselattribute benutzt werden. Dieses Objekt gibt es bei Oracle schon länger und ist nun im neuen SQL2003-Standard enthalten. Andere DBMS bieten zum Teil andere, nicht standardisierte Möglichkeiten³⁶ (AUTO_INCREMENT-Option in MySQL), mit denen man fortlaufende Nummern erzeugen kann. Aufgrund der späten Standardisierung gibt es sehr unterschiedliche Lösungen für dieses Problem.

```
<CREATE SEQUENCE Anweisung> ::= 
  CREATE SEQUENCE Sequenzname
  [ INCREMENT BY Integer ]
  [ START WITH Integer ]
  [ MAXVALUE Integer | NOMAXVALUE ]
  [ MINVALUE Integer | NOMINVALUE ]
  [ CYCLE | NOCYCLE ]
  [ CACHE | NOCACHE ]
  [ ORDER | NOORDER ]:
```

Dabei bedeuten die einzelnen Angaben folgendes:

Tabelle 5.20

CREATE SEQUENCE-Optionen	
INCREMENT BY int	Bestimmt die ganze Zahl, um die eine SEQUENCE erhöht wird (Default 1)
START WITH int	Bestimmt die ganze Zahl, mit der die SEQUENCE startet
MAXVALUE int NOMAXVALUE	Obere Grenze einer aufsteigenden SEQUENCE bzw. sie hat keine obere Grenze
MINVALUE int NOMINVALUE	Untere Grenze bei absteigenden Sequenzen bzw. sie hat keine untere Grenze
CYCLE NOCYCLE	Bestimmt, ob nach Erreichen des MAXVALUE bzw. MINVALUE zyklisch weiter vergeben wird oder ob da Ende ist
CACHE int NOCACHE	Bestimmt, wie viele Schlüsselwerte für einen schnelleren Zugriff im Hauptspeicher bereitgestellt werden. Bei NOCACHE werden keine Werte vorgehalten.

Beispiel

```
CREATE SEQUENCE Kun_seq INCREMENT BY 1 START WITH 1
  NOMAXVALUE NOCYCLE CACHE 10;
```

Sequenzen haben zwei Pseudospalten:

NEXTVAL	Enthält die nächste verfügbare fortlaufende Nummer
CURVAL	Enthält die zuletzt verwendete fortlaufende Nummer

Sequenzen werden hauptsächlich zum Einfügen von Daten mittels der INSERT-Anweisung benutzt (vgl. Abschnitt 5.4.1).

Beispiel

```
INSERT INTO Kunden (Kun_Nr, Nachname)
VALUES (Kun_seq.NEXTVAL, 'Vogt');
```

Diese Anweisung erzeugt einen neuen Datensatz mit der Kun_Nr als fortlaufende Nummer in der Kundentabelle.

5.3.7 Weitere CREATE-Anweisungen

Neben diesen Grundobjekten gibt es noch eine ganze Liste weiterer Datenbankobjekte, auf die an dieser Stelle nicht genauer eingegangen werden kann. Sie werden zur Übersicht tabellarisch aufgeführt. Der interessierte Leser sei z.B. auf die Oracle-Originalliteratur³⁷ oder die MySQL-Reference³⁸ verwiesen.

35 vgl. [ORACLE SQL 2005]

36 auch einen eigenen Datentyp in INFORMIX und MS ACCESS

37 vgl. [ORACLE SQL 2005]

38 vgl. [MySQL 2006]

Tabelle 5.21

Übersicht weiterer Datenbankobjekte	
USER	Datenbankbenutzer
TABLESPACE	Zusammenfassung von Tabellen, die physikalisch gemeinsam gespeichert werden (vgl. Kapitel 9)
CLUSTER	Verbindung von Tabellen, für die eine besondere Speicherstruktur vereinbart werden soll
DATABASE	Komplette Datenbank
SYNONYM	ALIAS-Name für eine Tabelle, Sicht oder Sequenz
ROLE	Zusammenfassung von Benutzerrechten
DIRECTORY	Pfadbezeichnung, in der externe Daten gespeichert werden (für BLOBS, Grafiken, lange Texte etc.)
FUNCTION	Gespeicherte PL/SQL- oder PSM-Funktion (vgl. Abschnitt 7.1 und 7.2)
PROCEDURE	Gespeicherte PL/SQL- oder PSM-Prozedur (vgl. Abschnitt 7.1 und 7.2)
TRIGGER	aktive Regeln (vgl. Abschnitt 7.3)

5.3.8 Die DROP-Anweisung

Mit der DROP-Anweisung können Datenbankobjekte wieder aus der Datenbank entfernt werden, sofern sie vorher angelegt wurden:

```
<DROP Anweisung> ::= DROP / TABLE Tabellename [CASCADE CONSTRAINTS]
| VIEW Sichtname
| INDEX Indexname
| SEQUenz Sequenzname;
```

Mit der DROP TABLE-Anweisung werden auch alle Indizes, Integritätsbedingungen TRIGGER und Sichten, die auf der Tabelle aufbauen, sowie die Daten selbst gelöscht. Tabellen können normalerweise nur dann gelöscht werden, wenn keine Fremdschlüsselbedingungen mehr existieren. Die Option CASCADE CONSTRAINTS ist eine Oracle-Erweiterung, die auch alle Fremdschlüsselbedingungen entfernt, die sich auf die Tabelle beziehen. Somit muss keine spezielle Reihenfolge mehr beim Löschen von Tabellen eingehalten werden. Will man CONSTRAINTS löschen, ohne die Tabelle selbst zu löschen, muss man die ALTER-Anweisung (vgl. Abschnitt 5.3.10) verwenden.

Beispiel

```
DROP TABLE Test;
DROP TABLE Teile CASCADE CONSTRAINTS;
```

5.3.9 Die RENAME-Anweisung

Die RENAME-Anweisung ändert den Namen einer Tabelle. Die aktuelle Tabelle darf dabei nicht in einer VIEW verwendet werden und keine CONSTRAINTS oder Datenbanktrigger (vgl. Kapitel 7) aufweisen. Ansonsten weist das Datenbanksystem die RENAME-Anweisung mit einer Fehlermeldung zurück.

```
<RENAME Anweisung> ::= RENAME alter Tabellename TO neuer Tabellename;
```

Diese Anweisung gehört nicht zum SQL2003-Sprachumfang und ist auch nicht in allen Systemen verfügbar:

Tabelle 5.22

Verfügbarkeit der RENAME-Anweisung

SQL2003	Oracle	MySQL
RENAME	X	X

Beispiel

```
RENAME Teile TO Teile_neu;
```

5.3.10 Die ALTER TABLE-Anweisung

Mit der ALTER TABLE-Anweisung³⁹ können nachträglich Spalten in Tabellen verändert, gelöscht oder hinzugefügt werden. Insbesondere lassen sich auch Spalten- und Tabellenbedingungen nachträglich bearbeiten. MySQL weicht hier ziemlich stark vom Standard ab, die Syntax wird im Abschnitt 5.3.12 behandelt. Oracle verlässt nur bei einem Schlüsselwort den Standard, dort heißt es dann MODIFY (<Spaltendefinition>) statt ALTER und setzt die <Spaltendefinition> jeweils in runde Klammern: (<Spaltendefinition>).

```
<ALTER TABLE Anweisung> ::=
ALTER TABLE Tabellename
/ ADD / (<Spaltendefinition> | <Tabellenbedingung> | CONSTRaINT Constraintname )
| DROP / Spaltenname
| ALTER <Spaltendefinition> ;
```

³⁹ Die ALTER-Table Anweisung gehörte noch nicht zum Sprachumfang von SQL1, sondern wurde erst in den Sprachumfang von SQL2 aufgenommen.

Beispiele

- Diese Anweisung fügt eine neue Spalte zur Teiletabelle hinzu.

```
ALTER TABLE Teile ADD (Preis NUMERIC);
```

- Diese Anweisung vergrößert die maximale Länge des Attributs Bezeichnung auf 70 Zeichen. Mit der Oracle MODIFY-Klausel können keine Datentypumwandlungen vorgenommen werden.

```
ALTER TABLE Teile MODIFY (Bezeichnung VARCHAR(70));
```

- Diese Anweisung entfernt die Spalte „Art“ aus der Tabellendefinition der Teiletabelle. Hier setzt Oracle gegenüber dem Standard den Spaltennamen in runde Klammern.

```
ALTER TABLE Teile DROP (Art);
```

- Diese Anweisung legt einen Primärschlüssel für die Auftragstabelle fest.

```
ALTER TABLE Auftraege ADD CONSTRAINT Auf_Pk PRIMARY KEY (AuftragsNr);
```

- Diese Anweisung legt einen Fremdschlüssel für die Artikeltabelle fest, der auf die Spalte TNr in der Teiletabelle verweist. Fremdschlüssel werden in der Praxis gerne mit ALTER-Anweisungen angelegt, weil man dann keine Reihenfolge bei den CREATE TABLE-Anweisungen beachten muss. Legt man die Fremdschlüssel direkt mit dem CREATE TABLE-Befehl an, muss die referenzierte Master-Tabelle bereits angelegt sein.

```
ALTER TABLE Artikel ADD CONSTRAINT Art_T_Fk
    FOREIGN KEY (TNr) REFERENCES Teile (TNr);
```

- Diese ALTER-Anweisung löscht den zuvor angelegten CONSTRAINT wieder. Oracle kennt auch eine MODIFY-Option für Integritätsbedingungen. SQL2003 noch nicht.

```
ALTER TABLE Artikel DROP CONSTRAINT Art_T_Fk;
```

5.3.10.1 Regeln zum Ändern von Spalten

Die ALTER TABLE-Anweisung zum Ändern von Spalten unterliegt einigen Einschränkungen.⁴⁰

Man kann immer

- die Nachkommastellen einer NUMERIC-Spalte eines anderen numerischen Zahlen- typs erhöhen,
- Textspalten im CHARACTER- oder VARCHAR-Format vergrößern sowie
- die Anzahl der Ziffern in einer NUMERIC-Spalte erhöhen.

Falls die Spalte nur NULL-Werte hat, kann man zusätzlich

- die Nachkommastellen einer NUMERIC-Spalte reduzieren,
- Textspalten im CHARACTER- oder VARCHAR-Format reduzieren,
- die Anzahl der Ziffern in einer NUMERIC-Spalte reduzieren sowie
- den Datentyp der Spalte ändern.

Leider ist es im Standard nicht möglich, eine Spalte nachträglich umzubenennen. MySQL hat dazu eine eigene Syntax, die dies vorsieht. Das gilt auch für andere ALTER-Optionen, die unter MySQL etwas anders aussehen (vgl. Abschnitt 5.3.12). Oracle bietet für diesen Zweck im Rahmen des ALTER TABLE-Befehls eine RENAME COLUMN-Klausel an.

Beispiel

Die Spalte Bestand der Tabelle Teile wird umbenannt in Bestandssumme:

```
ALTER TABLE Teile
RENAME COLUMN Bestand TO Bestandssumme;
```

Analog zur CREATE-Anweisung können andere Datenbankobjekte mit der ALTER- Anweisung nachträglich verändert werden, wie eine Sicht, ein Index, eine Prozedur, eine Funktion oder ein Benutzer. Die Anweisungen unterscheiden sich wieder sehr stark je nach Hersteller, so dass auf die Originalliteratur der einzelnen Hersteller verwiesen wird.

5.3.11 Oracle-Besonderheiten in der SQL-DDL-Syntax

Erst einmal lässt sich sagen, dass es große Unterschiede bei den Datentypen gibt, da intern nur die Datentypen CHAR, VARCHAR2, DATE und NUMBER neben den BLOBS benutzt werden. Die BLOBS werden mit PL/SQL-Prozeduren bearbeitet, wie in Abschnitt 7.1.2 beschrieben. Große Unterschiede bestehen auch bei den Single-Row-Funktionen, die in diesem Buch nur zu einem Bruchteil aufgeführt werden können. Es gibt eine Vielzahl von Objekten, die mit CREATE/ALTER/DROP in der Datenbank angelegt, geändert und entfernt werden können, aber hier nicht beschrieben sind. Eine Ausnahme ist die MATERIALIZED VIEW, die im Abschnitt 5.3.5 aufgenommen wurde. Bei der DROP- TABLE-Anweisung gibt es die sehr nützliche Möglichkeit, mit CASCADE CONSTRAINTS alle störenden CONSTRAINTS zu entfernen. Große Tabellen können auch partitioniert werden, d.h. die Tabellen werden zeilenweise aufgeteilt und mittels der CREATE TABLE-Anweisung auf verschiedene Partitionen verteilt. Außer den Standard- Index-Typen gibt es noch einen Bitmap-Index für große Datenbestände, die sich selten ändern. Wer sich für diese Einzelheiten interessiert, sei wieder auf [Loney 2005] oder die Oracle-Dokumentation verwiesen. Der ALTER-Befehl weist ein paar kleine syntaktische Abweichungen auf. So heißt die ALTER-Option hier MODIFY und die Spaltendefinition bzw. -änderung nach dem ADD bzw. MODIFY wird in runde Klammern gesetzt. Funktional ist insbesondere die Erweiterung um die De-/Aktivierung von erzeugten TRIGGERS und CONSTRAINTS interessant (vgl. Abschnitt 5.3.2 und 7.3).

<i><Aktivierungsklausel beim ALTER> ::=</i> <i>[/ ENABLE DISABLE] [/ ALL TRIGGERS</i> <i> UNIQUE (Spaltenname [, Spaltenname]...)</i> <i> PRIMARY KEY</i> <i> CONSTRAINT Constraintname /]</i>

40 vgl. [Loney 2005, S. 363]

5.3.12 MySQL-Besonderheiten in der SQL-DDL-Syntax

MySQL hat wie alle kommerziellen Datenbanken bei den Datentypen zusätzlich eigene Typen, die nicht dem Standard entsprechen. Diese sind im Abschnitt 5.2.1 beschrieben. Große Unterschiede findet man bei der Formatierung von Datumsfeldern, was eine unterschiedliche Syntax der INSERT-Anweisungen zur Folge hat, da andere Datumsfunktionen und eine fixe Formatierung der Gestalt „YYYY-MM-DD“ verwendet werden müssen. Auch die Behandlung von BLOBS ist sehr verschieden (vgl. Kapitel 6).

Die wichtigste zusätzliche Option ist die Festlegung von unterschiedlichen Speichermaschinen (Storage Engine) in der CREATE TABLE-Anweisung. Einen Überblick über die Charakteristika von Speichermaschinen finden Sie in Tabelle 1.3 im Abschnitt 1.9.3. Beispiele für ihre Verwendung bieten die Kapitel 7.3 und 8. Für die Themen hier in diesem Buch sind die beiden Speichermaschinen MyISAM (DEFAULT) und InnoDB ausreichend – mit dem für uns relevanten Unterschied, dass nur InnoDB über ein Transaktionskonzept verfügt (vgl. Abschnitt 8.6). Die ENGINE-Option wird hinter der runden Klammer unmittelbar vor dem abschließenden Semikolon beim CREATE TABLE eingefügt.

```
<Speichermaschinen Definition beim CREATE TABLE> ::=  
    ENGINE='Speichermaschine'
```

Natürlich sind auch die Single-Row-Funktionen MySQL-spezifisch. Besonders nützliche Klassen von Funktionen sind diejenigen zur Datenverschlüsselung (vgl. Abschnitt 7.2.2).

Was weiterhin sofort ins Auge fällt, ist die unterschiedliche Behandlung von fortlaufenden Nummern. Während Oracle und der SQL-Standard SEQUENCES vorsieht, hat MySQL hier ein AUTO_INCREMENT als Spaltenattribut⁴¹. Das Fehlen von Sequenzen wirkt sich auch auf eine unterschiedliche Behandlung der AUTO_INCREMENT-Spalten in INSERT-Statements aus (vgl. Abschnitt 5.4.1). Dort finden Sie auch ein Beispiel.

Neben der bekannten SQL-Funktionalität des ALTER TABLE-Befehls gibt es noch eine Vielzahl an Zusatzfunktionen, von denen hier nur drei herausgegriffen werden. Man kann eine Tabelle umbenennen (anstelle von RENAME unter Oracle) oder auch eine Spalte umbenennen und ihr gleichzeitig einen neuen Datentyp zuweisen. Fehlt der Datentyp, wird die Spalte nur umbenannt. Für die Deaktivierung von Integritätsbedingungen steht die Option ENABLE/DISABLE KEYS zur Verfügung.

```
<ALTER TABLE Anweisung> ::=  
    ALTER TABLE Tabellename | RENAME Neuer_Tabellename  
        | CHANGE Spaltenname Neuer_Spaltenname [ <Datentyp> ]  
        | | ENABLE | DISABLE | KEYS |;
```

⁴¹ vgl. [Buchmann 2005, S. 83 ff.]

5.4 Die Datenmanipulationssprache (DML, Data Manipulation Language)

Zu diesem Sprachbestandteil von SQL gehören Anweisungen zum Einfügen (INSERT), Ändern (UPDATE) und Löschen (DELETE) von Daten.

5.4.1 Die INSERT-Anweisung

Mit der INSERT-Anweisung werden Daten in Tabellen eingefügt.

```
<INSERT Anweisung> ::=  
    INSERT INTO Tabellename [ ( Spaltenname [, Spaltenname ]... ) ]  
        / VALUES ( Konstante [, Konstante ]... ) | <Anfrageausdruck> |;
```

Anmerkungen

- Die Anzahl und Datentypen der Spalten hinter dem Tabellennamen müssen mit der Konstantenliste bzw. dem Ergebnis einer beliebig komplexen SELECT-Anweisung, wie wir sie im Abschnitt 5.5.11 beschrieben haben, übereinstimmen.
- Wenn die Spaltennamen fehlen, müssen alle Spalten der Tabelle in der Reihenfolge der Tabellenspalten mit Werten versorgt werden. Werden Spaltennamen angeführt, so ist die Reihenfolge beliebig.

Beispiele

```
INSERT INTO Teile (TNr, Typ, Bezeichnung)  
VALUES (38, 'Baugruppe', 'Dynamo');
```

```
INSERT INTO Lagerbestand  
VALUES (3, 58, 899, CURRENT_DATE);
```

```
INSERT INTO Abteilungen  
VALUES (2,2,'Produktion','Lindlar');
```

```
INSERT INTO Kunden (Kun_Nr, Nachname)  
SELECT Ang_Nr, Nachname FROM Angestellte;
```

Die letzte Anweisung trägt alle Angestellten auch in die Kundentabelle ein. Alle Spalten, die in der Spaltenliste nicht auftauchen, müssen NULL-Werte erlauben und werden mittels dieser Anweisung auch mit NULL-Werten gefüllt.

Falls eine Tabelle eine Spalte mit fortlaufender Nummer hat, benötigt man dazu unter Oracle und Standard-SQL eine Sequenz. Mit der im Abschnitt 5.3.6 definierten Sequenz Kun_seq kann man z.B. einen Kunden einfügen mit

```
INSERT INTO Kunden (Kun_Nr, Nachname)  
VALUES (Kun_seq.NEXTVAL, 'Vogt');
```

Unter MySQL wird die fortlaufende Nummer als AUTO_INCREMENT vereinbart:

```
CREATE TABLE Kunden  
(Kun_Nr INT AUTO_INCREMENT PRIMARY KEY,  
NACHNAME VARCHAR(50));
```

Die INSERT-Anweisung braucht die Kun_Nr nicht zu berücksichtigen, da die Spalte automatisch gefüllt wird, indem die Kun_Nr bei jedem INSERT um 1 erhöht wird.

```
INSERT INTO Kunden (Nachname) VALUES('Vogt');
```

5.4.2 Die UPDATE-Anweisung

Mit der UPDATE-Anweisung können Mengen von Daten oder einzelne Datensätze in Tabellen verändert werden.

```
<UPDATE Anweisung> ::= UPDATE Tabellename
    SET <Wertzuweisung> [, <Wertzuweisung> ]...
    [ WHERE <Suchbedingung> ];
<Wertzuweisung> ::= Spaltenname = / <Ausdruck> | NULL | <Anfrageausdruck> /
```

Die Suchbedingung ist wie in der SELECT-Anweisung definiert (vgl. Abschnitt 5.5.5). Die Wertzuweisung lässt einen NULL-Wert zu, einen Ausdruck wie bei der Default-Option der CREATE TABLE-Anweisung (vgl. Abschnitt 5.3.1), oder einen Anfrageausdruck, wie er in Abschnitt 5.5.11 erläutert wird. Der Anfrageausdruck darf hier keine Datensatzmenge als Ergebnis liefern, sondern nur einen Datensatz. Je nachdem, ob in der SET-Klausel nur eine einzelne oder mehrere Spalten geändert werden, können in den Anfrageausdruck eine oder entsprechend auch mehrere Spalten selektiert werden (vgl. [ANSI SQL 2003b] und [Oracle SQL 2005]). Hier haben wir uns auf die Syntax für Wertänderungen von einer Spalte beschränkt.

Beispiele

- Ändern Sie in der Tabelle „Artikel“ alle Bezeichnungen „Steppenwolf TAO“ in „Steppenwolf TAO 2“!

```
UPDATE Artikel
    SET Bezeichnung = 'Steppenwolf TAO 2'
    WHERE Bezeichnung = 'Steppenwolf TAO';
```

Auch bei der UPDATE-Anweisung ist es möglich, wie bei der SELECT-Anweisung eine vollständige Suchbedingung in der WHERE-Klausel unterzubringen.

- Allen Angestellten der Abteilung 2 werden 10 % vom Gehalt abgezogen.

```
UPDATE Angestellte
    SET Gehalt = Gehalt * 0.9
    WHERE Abt_Nr = 2;
```

- In der Tabelle „Lagerbestand“ wird für alle Datensätze der Zeitstempel auf das Systemdatum gesetzt!

```
UPDATE Lagerbestand SET Zeitstempel = CURRENT_DATE;
```

Nicht zulässig sind in der UPDATE-Anweisung das gleichzeitige Ändern von Daten in mehreren Tabellen oder das gleichzeitige Ändern von Tabellen, die durch einen JOIN miteinander verbunden sind. Welche Probleme damit verbunden sein können und wie man sie lösen kann, ist sehr schön in Fritze⁴² dargestellt.

Die MERGE-Anweisung ist in den Standard SQL2003 aufgenommen worden und fügt Daten in Tabellen ein oder ändert sie, je nachdem, ob die Daten schon als Datensatz vorhanden sind oder nicht. MERGE existiert in Oracle seit der Version 9, unter MySQL gibt es diese Anweisung nicht.

5.4.3 Die DELETE-Anweisung

Die DELETE-Anweisung dient zum Löschen von Datensätzen aus Tabellen. Die Tabellendefinitionen bleiben dabei erhalten.

```
<DELETE Anweisung> ::= DELETE FROM Tabellename
    [ WHERE <Suchbedingung> ];
```

Alle Datensätze, für die die Suchbedingung TRUE ist, werden gelöscht. Die Suchbedingung ist wie in der SELECT-Anweisung definiert (vgl. Abschnitt 5.5.5).

Beispiele

```
DELETE FROM Angestellte;
```

löscht alle Daten in der Tabelle „Angestellte“.

```
DELETE FROM Angestellte WHERE Nachname LIKE 'M%';
```

löscht alle Angestellten, deren Nachname mit M beginnt.

5.4.4 Oracle-Besonderheiten in der SQL-DML-Syntax

Die INSERT Anweisung lässt über den Standard hinaus ein INSERT ALL zum Einfügen mehrerer Datensätze und ein INSERT FIRST mit einer WHEN-Klausel zu, die Daten nur einfügt, falls die WHEN-Klausel erfüllt ist⁴³.

Die TRUNCATE-Anweisung arbeitet ähnlich wie die DELETE-Anweisung, allerdings ohne eine WHERE-Klausel und ohne die Möglichkeit, eine Transaktion mit ROLLBACK zurückzurollen. Dadurch wird die Anweisung erheblich schneller.

⁴² vgl. [Fritze et al. 2002, S. 152 ff.]

⁴³ vgl. <http://www.oracle-base.com/articles/9i/SQLNewFeatures9i.php>, 10.11. 2006 und [Rischert 2004, S. 446]