*Gunter Saake*
*Kai-Uwe Sattler*
*Andreas Heuer*

# Datenbanken

## Konzepte und Sprachen

**Sechste Auflage**

Dieses pdf-Kapitel ist eine kostenlose Ergänzung zum oben genannten Buch, das 2018 bei MITP erschienen ist.

Bitte beachten: Verweise auf Umgebungen B-X beziehen sich auf Teile innerhalb dieses Download-Kapitels. Verweise, die mit Kapitelnummern beginnen, wie 11-31, beziehen sich auf Umgebungen innerhalb des obigen Lehrbuches. Verweise auf Seiten vor 738 beziehen sich ebenfalls auf das zugrundeliegende Lehrbuch.

Literaturhinweise in diesen pdf-Kapiteln beziehen sich auf ein erweitertes Literaturverzeichnis zum Lehrbuch, das auch als kostenlose Ergänzung an derselben Stelle zum Download bereitsteht.

**Bibliografische Information der Deutschen Nationalbibliothek**
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Bei der Herstellung des Werkes haben wir uns zukunftsbewusst für umweltverträgliche und wiederverwertbare Materialen entschieden. Der Inhalt ist auf elementar chlorfreiem Papier gedruckt.

ISBN 978-3-95845-776-8
6. Auflage 2018

www.mitp.de
E-Mail: mitp-verlag@sigloch.de

Telefon: +49 7953/7189-079
Telefax: +49 7953/7189-082



Lektorat: Sabine Janatschek, Ernst-Heinrich Pröfener
Sprachkorrektorat: Jürgen Dubau, Astrid Langen
Covergestaltung: Christian Kalkert, www.kalkert.de
Bildnachweis Cover: iStock.com/Shawn Hempel | fotolia.com/karpenko_ilia
Satz: Gunter Saake, Magdeburg; Kai-Uwe Sattler, Ilmenau; Andreas Heuer, Rostock
Druck: Westermann Druck Zwickau GmbH

## B.5 Tutorial D

Trotz der weiten Verbreitung und Akzeptanz von SQL gibt es eine ganze Reihe von Kritikpunkten. Hierzu zählen das Fehlen bestimmter Konstrukte bzw. Operatoren, die teilweise umständliche Syntax und insbesondere der Unterschied des SQL-Modells im Vergleich zum Codd'schen Relationenmodell, wie u.a. Relationen (Tabellen) als Multimengen und Nullwerte.

Date und Darwen haben daher in ihrem Manifesto [DD00] mit *Tutorial D* eine relationale Sprache (genau genommen: eine Menge von Anforderungen an relationale Sprachen) vorgeschlagen.

Leider gibt es abgesehen von einigen Lehrprojekten noch keine ernstzunehmende Implementierung von Tutorial D. Trotzdem wollen wir in diesem Abschnitt Tutorial D kurz vorstellen, da die Sprache einige interessante Features bietet. Wir beziehen uns dabei auf die in [Dat05] behandelte Version.

Tutorial D wurde primär zu Lehrzwecken eingeführt und basiert konsequent auf den Basiskonzepten des Relationenmodells und der Relationenalgebra. Insofern unterscheidet sich die Sprache deutlich von SQL, QUEL, QBE und Datalog, die im diesem Abschnitt zugrundeliegenden Lehrbuch an anderer Stelle erläutert wurden: alle anderen Sprachen hatten den Tupel- oder Bereichskalkül als Ausgangspunkt (SQL) oder Grundlage.

### B.5.1 Datentypen

Tutorial D umfasst Konstrukte zur Datendefinition und -manipulation, Anfrageoperatoren sowie imperative Elemente. Ein wichtiges Merkmal ist dabei die saubere Trennung von Relationen und Typen einerseits sowie Relationenwerten und Relationenvariablen andererseits. Während etwa in SQL der Begriff der Tabelle sowohl für die Relation (Menge der Tupel) als auch den Typ steht, wird dies in Tutorial D unterschieden. So gibt es einen Typkonstruktor **`relation`**, mit dessen Hilfe ein Relationentyp erzeugt werden kann. Genau wie die Basisdatentypen **`integer`** oder **`char`** kann dieser Relationentyp einen Wert haben: eben eine Relation. In gleicher Weise lassen sich Variablen einführen, die einen Typ haben und an die Werte gebunden werden können.

**◂Beispiel B-30▸** Wir definieren eine Variable `WEINE` mit einem Relationentyp, der unseren bisher verwendeten SQL-Tabellen entspricht:

```
var WEINE base
   relation { WeinID integer,
      Name char,
      Farbe char,
      Jahrgang integer,
      Weingut char,
      Preis real }
   key { WeinID }
   foreign key { Weingut } references ERZEUGER ;
```

Die Variablendefinition wird hier durch **var** eingeleitet, **base** gibt an, dass es sich um eine Basisrelation (in Abgrenzung zu Sichten, die mit **view** definiert werden) handelt. Das Schlüsselwort **key** leitet die Schlüsseldefinition ein, **foreign key** entsprechend die Definition des Fremdschlüssels. □

Es sei angemerkt, dass WEINE nicht der Typname ist, sondern eine Variable: der Name des Relationentyps ist das Relationenschema!

Tutorial D ist nicht auf die üblichen Basistypen wie **integer**, **char** usw. beschränkt, sondern erlaubt auch benutzerdefinierte Typen. Darüber hinaus können Attribute auch relationenwertig sein, d.h. es werden geschachtelte Relationen unterstützt. Schließlich sind auch Tupeltypen möglich:

**◂Beispiel B-31▸** Wir führen eine Variable T eines Tupeltyps mit den Komponenten des obigen Relationentyps ein.

```
var T tuple { WeinID integer,
      Name char,
      Farbe char,
      Jahrgang integer,
      Weingut char,
      Preis real };
```

Über den Zuweisungsoperator := kann nun ein Tupelwert an diese Variable gebunden werden:

```
T := tuple { WeinID 3456, Name 'Zinfandel', Farbe 'Rot',
   Jahrgang 2004, Weingut 'Helena', Preis 5.99 };
```

Das Schlüsselwort **tuple** dient hier in Verbindung mit dem Typnamen als Tupelselektor. □

Mithilfe der Tupelselektoren kann nun auch eine Relation konstruiert werden:

**◂Beispiel B-32▸** Der Variable WEINE soll eine Relation mit mehreren Tupeln zugewiesen werden:

```
WEINE := relation {
   tuple { WeinID 2168, Name 'Creek Shiraz', Farbe 'Rot',
      Jahrgang 2003, Weingut 'Creek', Preis 7.99 },
   tuple { WeinID 3456, Name 'Zinfandel, Farbe 'Rot',
      Jahrgang 2004, Weingut 'Helena', Preis 5.99 },
   ...};
```

□

Zu beachten ist der Unterschied zwischen einem Tupel und einer Relation, die nur ein Tupel enthält: Letztere ist eine einelementige Menge und somit sind Tupel und Relation Instanzen von verschiedenen Typen.

### B.5.2 Anfrageoperatoren

Die Operationen der Relationenalgebra werden in Tutorial D direkt unterstützt. So gibt es für Projektion und Selektion ebenso eigene Operatoren wie für den (natürlichen) Verbund.

*Projektion.* Die Projektion $\pi$ wird durch den Restrict-Operator durchgeführt, bei dem die in der Ergebnisrelation gewünschten Attribute aufgeführt werden. Der Algebraausdruck

$$\pi_{\texttt{Name,Weingut}}(\texttt{WEINE})$$

wird in Tutorial D wie folgt formuliert, wobei `WEINE` die oben eingeführte Relationenvariable ist:

```
WEINE { Name, Weingut }
```

Alternativ ist auch eine Angabe der zu unterdrückenden Attribute möglich:

```
WEINE { all but WeinID }
```

wobei alle Attribute mit Ausnahme von `WeinID` ausgegeben werden.

*Selektion.* Bei der Selektion ist wie üblich eine Bedingung anzugeben. Dementsprechend wird der Algebraausdruck

$$\sigma_{\texttt{Farbe}=\text{'Rot'}}(\texttt{WEINE})$$

in dieser Form dargestellt:

```
WEINE where Farbe = 'Rot'
```

*Verbund.* Für den natürlichen Verbund gibt es einen eigenen **join**-Operator, so dass der Algebraausdruck

$$\texttt{WEINE} \bowtie \texttt{ERZEUGER}$$

direkt als

```
WEINE join ERZEUGER
```

notiert werden kann. ERZEUGER ist hier eine Relationenvariable mit einem Typ, der dem Relationenschema aus Abschnitt 5.1 entspricht. Neben dieser Infixnotation ist auch eine Präfixschreibweise möglich, die insbesondere bei Mehrwegeverbunden bequemer ist:

```
join { WEINE, ERZEUGER }
```

Kombiniert werden diese Operatoren einfach durch Klammerung, wie das folgende Beispiel demonstriert.

**◂Beispiel B-33▸** Mit der Anfrage sollen die Namen aller kalifornischen Rotweine ermittelt werden:

```
((WEINE where Farbe = 'Rot')
   join
(ERZEUGER where Region = 'Kalifornien')) { Name }
```

□

*Weitere Projektionen.* Auch eine Umbenennung einzelner Attribute über den **rename**-Operator sowie die Erweiterung der Relation um berechnete Attribute mithilfe des **extend**-Operators ist möglich. Dies steht im Gegensatz zu SQL, wo diese Operationen alle Teil der **select**-Klausel sind.

**◂Beispiel B-34▸** Dieses Beispiel soll die Notation in SQL und Tutorial D gegenüberstellen. Die SQL-Anfrage

```
select WeinID, Name, Farbe, 2007 - Jahrgang as Alter,
   Weingut as Hersteller, Preis
from WEINE
where 2007-Jahrgang > 2
```

wird in Tutorial D wie folgt umgesetzt:

```
(extend WEINE add (2007-Jahrgang as Alter))
   rename (Weingut as Hersteller) where Alter > 2
```

□

*Mengenoperationen.* In Tutorial D sind natürlich auch die Mengenoperationen **union**, **minus** und **intersect** verfügbar, die ebenfalls in Infixnotation eingesetzt werden.

**◂Beispiel B-35▸** Die Umsetzung von Beispiel 5-12 auf Seite 123 mit der Anfrage

$$\texttt{WEINLISTE} \cup \beta_{\texttt{Name}\to\texttt{Wein}}\texttt{EMPFEHLUNG}$$

in Tutorial D ist unter Verwendung von zwei Relationenvariablen einfach:

```
WEINLISTE union (EMPFEHLUNG rename (Wein as Name))
```

□

*Gruppierung und Aggregation.* Eine der Gruppierung und Aggregation ähnliche Operation ist mit dem **summarize**-Operator möglich, der folgende Syntax hat:

```
summarize r per ( s ) add (summary as a)
```

Hierbei sind $r$ und $s$ Relationen und der Typ von $s$ ist eine Projektion von $r$ – demzufolge kann auch gelten $r = s$. Dieser Operator liefert eine Relation, deren Typ dem Typ von $s$, erweitert um das Attribut $a$, entspricht. Diese Ergebnisrelation enthält alle Tupel $t \in s$, erweitert um die $a$-Werte, die durch Anwendung der *summary*-Funktion berechnet werden. Hierbei handelt es sich im Prinzip um die bekannten Aggregatfunktionen **count**, **min**, **max**, **sum**, **avg** etc. Diese existieren auch als Varianten mit Duplikateliminierung (**countd**, **sumd**, **avgd**). Betrachten wir hierzu ein Beispiel.

**◂Beispiel B-36▸** Wie in Beispiel 11-18 auf Seite 306 soll in der Relation `WEINE` die Anzahl der Rot- bzw. Weißweine ermittelt werden. Hierzu muss die **summarize**-Operation auf die Relation `WEINE` angewendet werden, wobei dies für die Projektion auf `Farbe` auszuführen ist (**per**-Klausel). Weiterhin muss das Attribut `Anzahl` hinzugefügt werden, das mithilfe der **count**-Funktion berechnet wird.

```
summarize WEINE per (WEINE { Farbe } )
   add (count() as Anzahl)
```

□

Neben den beim **summarize**-Operator verwendeten *summary*-Funktionen kennt Tutorial D noch echte Aggregatfunktionen, die die gleiche Bezeichnung haben, jedoch auch als skalare Ausdrücke eingesetzt werden können.

**◂Beispiel B-37▸** Betrachten wir hierzu eine Anwendung von **count** als Aggregatfunktion zur Bestimmung der Anzahl der Rotweine.

```
var num integer;
num := count(WEINE where Farbe = 'Rot');
```

□

*Entschachtelung.* Da Tutorial D grundsätzlich auch relationenwertige Attribute zulässt, werden entsprechende Operationen zum Entschachteln bzw. Schachteln benötigt. In Abschnitt B.3.3.1 haben wir dazu die Operatoren $\mu$ und $\nu$ eingeführt, die in Tutorial D **ungroup** und **group** heißen.

**◂Beispiel B-38▸** Gegeben seien die beiden nachfolgend dargestellten Relationen:

**W**

| Weingut | Wein |
|---|---|
| Creek | Creek Shiraz |
| Creek | Pinot Noir |
| Helena | Zinfandel |
| Helena | Pinot Noir |
| Müller | Riesling |
| Chateau La Rose | Grand Cru |

**E**

| Weingut | Weine |
|---|---|
| Creek | Creek Shiraz<br>Pinot Noir |
| Helena | Zinfandel<br>Pinot Noir |
| Müller | Riesling |
| Chateau La Rose | Grand Cru |

Mit der Anfrage

```
W group ( { Wein } as { Weine } )
```

wird aus der Relation W die Relation E erzeugt. Mit der Anfrage

```
E ungroup ( Weine )
```

wird die Relation E entschachtelt und entspricht somit der Relation W. □

Mit diesen Beispielen haben wir nur einen kurzen Überblick zu den Anfrageoperatoren gegeben. Dennoch sollte die Nähe zur Relationenalgebra deutlich geworden sein.

## B.5.3 Änderungsoperationen

Änderungen an Relationen sind aufgrund der Verfügbarkeit des relationalen Zuweisungsoperators := und der Relationenvariablen im Prinzip ohne zusätzliche Konzepte möglich. Wie in Abschnitt 5.3.2 vorgestellt, kann beispielsweise das Einfügen durch Vereinigung der Relation mit dem neuen Tupel erfolgen. Dennoch stehen in Tutorial D die üblichen **insert**-, **update**- und **delete**-Operationen zur Verfügung, die grundsätzlich auf Relationen arbeiten.

**◂Beispiel B-39▸** Als Beispiel betrachten wir das Einfügen eines neues Weingutes in die ERZEUGER-Relation zunächst mithilfe der Zuweisung:

```
ERZEUGER := ERZEUGER union ( relation {
   tuple { Weingut 'Chateau Lafitte',
      Anbaugebiet 'Medoc', Region 'Bordeaux' } } );
```

Die äquivalente **insert**-Anweisung lautet dann:

```
insert ERZEUGER (relation {
   tuple { Weingut 'Chateau Lafitte',
      Anbaugebiet 'Medoc', Region 'Bordeaux' } } );
```

Am Typkonstruktor **relation** ist ersichtlich, dass die Eingabe für **insert** immer eine Relation sein muss. □

Ähnliches gilt auch für die anderen Änderungsanweisungen. Daher betrachten wir nur noch ein Beispiel für die Anwendung von **delete** und **update** und verzichten auf die Darstellung der erweiterten Form mit dem Zuweisungsoperator.

**◂Beispiel B-40▸** Aus unserer Relation WEINE sollen alle Weißweine gelöscht werden:

```
delete WEINE where Farbe = 'Weiß';
```

Außerdem soll der Preis des Weines mit der WeinID geändert werden:

```
update WEINE where WeinID = 3456 (Preis := 7.99)
```

□

### B.5.4 Constraints

Genau wie SQL erlaubt auch Tutorial D die Definition von Integritätsbedingungen. Neben den bereits vorgestellten Schlüssel- und Fremdschlüsselbedingungen sind dies Typ-Constraints und Datenbank (Relationen)-Constraints.

Typ-Constraints ermöglichen die Einschränkung des Wertebereichs und sind damit mit den Domain-Constraints vergleichbar.

**◂Beispiel B-41▸** Betrachten wir die Definition eines Datentyps Alter z.B. für das Alter eines Weines.

```
type Alter possrep {
   A integer constraint A ≥ 0 and A ≤ 20 }
```

Dieser Typ basiert auf **integer**-Werten im Wertebereich $0 \ldots 20$.

□

Das Schlüsselwort **possrep** steht hierbei für „possible representation“ und gibt die für den Nutzer sichtbare Repräsentation des Typs (im obigen Beispiel

**integer**). Diese kann sich durchaus von der internen (physischen) Repräsentation der Werte unterscheiden, ist jedoch für die Nutzung des Typs notwendig.

Datenbank-Constraints schränken die möglichen Zustände der Datenbank ein und sind mit den **assertion**-Bedingungen von SQL vergleichbar.

### B.5.5 Übungsaufgaben

**Übung B-1** Setzen Sie die Anfrage aus Aufgabe 5-5 in Tutorial D um. □