Ludwig-Maximilians-Universität München
Institut für Informatik
Priv.-Doz. Dr. R. Hennicker
Matthias Hölzl

Sommersemester 1999

Einführung in die Informatik: Systeme und Anwendungen
Zusatzblatt 2: Integritätsbedingungen (vorläufige Version)

1. Einführung

Die in einer Datenbank gespeicherten Daten sollen in einem sinnvollen Zusammenhang mit der Wirklichkeit stehen. Z.B. ist es nicht sinnvoll ein Gewicht von -5kg in eine Tabelle einzutragen. Die Einhaltung derartiger Anforderungen wird durch sog. Ingegritätsbedingungen gewährleistet. Die meisten dieser Regeln sind datenbankspezifisch und drücken die Bedeutung der einzelnen Tabellen aus, man bezeichnet diese Regeln als *einfache Integritätsbedingungen* oder auch *Constraints*. Es gibt aber auch Ingegritätsbedingungen, die für alle Datenbanken gelten müssen. Die Einhaltung dieser Bedingungen wird als *referentielle Integrität* bezeichnet.

2. Einfache Ingeritätsbedingungen

Definiert man eine Tabelle `teile` durch die SQL-Anweisung

```
create table teile (teil_nr integer, gewicht integer)
```

so kann der Benutzer negative Gewichte eingeben. Um solche sinnlosen Einträge zu vermeiden gibt es die Möglichkeit die Werte, die die Spalte `gewicht` annehmen kann, auf positive ganze Zahlen zu beschränken. Dazu fügt man eine `check`-Klausel in die Anweisung ein:

```
create table teile (teil_nr integer, gewicht integer,
                    check (gewicht > 0))
```

Die `check`-Klausel bewirkt, dass bei jedem Einfügevorgang in die Tabelle `teile` und bei jeder Modifikation der Tabelle `teile` vom DBMS überprüft wird, ob der neue Wert der Spalte `gewicht` in der betreffenden Zeile (oder den betreffenden Zeilen) größer als 0 ist. Sollte das nicht der Fall sein, so wird das Kommando nicht ausgeführt und eine Fehlermeldung ausgegeben.

Ebenso könnte man durch eine `check`-Klausel der Form

```
create table teile (teil_nr integer, gewicht integer,
                    check (gewicht in (20, 50, 100, 250, 500)))
```

die in der Spalte `gewicht` vorkommenden Werte auf eine der angegebenen Zahlen beschränken.

3. Schlüssel

In SQL ist es möglich, dass mehrere Zeilen einer Tabelle für alle Spalten die gleichen Werte haben. Derartige Zeilen lassen sich nicht einzeln ansprechen, ändern oder löschen. Damit das DBMS gewährleisten kann, dass jede Zeile in einer Tabelle identifiziert werden kann, muss sie sich in mindestens einer Spalte von jeder anderen Zeile unterscheiden. In vielen Fällen gibt es einen in der Tabelle vorkommenden Wert, über den man jeden Datensatz eindeutig identifizieren kann, z.B. ist es sinnvoll in der Tabelle `teile` die Spalte `teil_nr` so zu wählen, dass jede Zeile eindeutig über die `teil_nr` angesprochen werden kann. Für die folgende Tabelle

spalte1	spalte2
1	1
1	2
2	2

lässt sich allerdings keine einzelne Spalte angeben, durch die sich die Zeilen identifizieren lassen, durch die Kombination der Werte für `spalte1` und `spalte2` lassen sich die einzelnen Datensätze aber auseinanderhalten. Für viele Anwendungen ist es nützlich, wenn man möglichst wenige Spalten zur Unterscheidung der einzelnen Zeilen heranziehen muss. Das führt zu der folgenden Definition:

Definition 3.1. Ein *Schlüsselkandidat* S für eine Tabelle T ist eine Menge von Spalten, die folgende Bedingungen erfüllt:

Eindeutigkeit: Zwei Zeilen von T unterscheiden sich in mindestens einer in S enthaltenen Spalte.

Irreduzibilität: Keine echte Teilmenge von S ist eindeutig, d.h., für jede echte Teilmenge S' von S gibt es mindestens zwei Zeilen, die für alle in S' enthaltenen Spalten übereinstimmen.

Ein Schlüsselkandidat ist also eine besonders „kleine“ Menge von Spalten, durch die man jede Zeile eindeutig identifizieren kann. Enthält ein Schlüsselkandidat mehr als eine Spalte, so spricht man manchmal von einem zusammengesetzten Schlüsselkandidaten.

Für jede Tabelle, die keine Zeilen doppelt enthält gibt es einen Schlüsselkandidaten: Die aus allen Spalten bestehende Menge ist eindeutig (denn es kommen keine Zeilen doppelt vor). Durch weglassen von überflüssigen Spalten kann man Irredzibilität erhalten.

Manche Tabellen haben nur einen Schlüsselkandidaten, andere Tabellen besitzen mehrere, aber in den meisten Fällen gibt es einen besonders einfachen oder „natürlichen“ Schlüsselkandidaten. Diesen bezeichnet man dann als den *Primärschlüssel,* alle anderen Schlüsselkandidaten nennt man *Alternativ-Schlüssel.*

In SQL wird eine Spalte durch Angabe des Schlüsselwortes `PRIMARY KEY` als Primärschlüssel definiert, soll also in der Definition

```
create table buecher(buch_nr char(4),
                     titel char(30), autor char(30));
```

die Spalte `buch_nr` als Primärschlüssel definiert werden, so schreibt man

```
create table buecher(buch_nr char(4) primary key,
                     titel char(30), autor char(30));
```

oder alternativ dazu

```
create table buecher(buch_nr char(4),
                     titel char(30), autor char(30),
                     primary key(buch_nr));
```

Indem man eine durch Kommata getrennte Liste von Spalten angibt kann man mit dieser letzten Notation auch zusammengesetzte Primärschlüssel definieren: Durch die Definition

```
create table buecher(isbn_nr char(10), exemplar_nr integer,
                     titel char(30), autor char(30),
                     primary key(isbn_nr, exemplar_nr));
```

wird der aus den zwei Spalten `isbn_nr` und `exemplar_nr` bestehende Primärschlüssel definiert. Analog verfährt man bei der Definition von Alternativschlüsseln, hier wird das Schlüsselwort `UNIQUE` verwendet:

```
create table buecher(isbn_nr char(10), exemplar_nr integer,
                     titel char(30), autor char(30),
                     primary key(isbn_nr, exemplar_nr),
                     unique(titel, autor));
```

Diese Definition erzeugt eine Tabelle `buecher` mit dem zusammengesetzten Primärschlüssel {`isbn_nr`, `exemplar_nr`} und den Alternativ-Schlüssel {`titel`, `autor`}

4. Referentielle Integrität

In einer Bibliotheksverwaltung könnte man die Kundendaten in einer Tabelle `kunden` speichern, die Daten der Bücher in einer Tabelle `buecher`. Im folgenden habe jeder Kunde der Bibliothek eine eigene Kundennummer und jedes in der Bibliothek vorhandene Buch eine eigene Inventarnummer. Beispielsweise könnten die `kunden`- und `buecher`-Tabellen die folgende Gestalt haben:

kunden_nr	name
24	Claudia Müller
31	Hans Huber
44	Nicole Kiefer

inventar_nr	titel
11	An Introduction to Database Systems
13	The SQL Standard
18	ANSI Common Lisp
19	ANSI Common Lisp
24	Einführung in die Volkswirtschaftslehre

In den meisten Datenbanken enthalten einige Tabellen Verweise auf andere Tabellen. Beispielsweise lässt sich ein Verleihvorgang in der Bibliotheksverwaltung durch eine Tabelle `verleih` repräsentieren, die eine Spalte für die Kundennummer des Entleihers und eine Spalte für die Inventarnummer des Buches enthält. Die Tabelle `verleih` enthält dann für jeden Verleihvorgang eine Zeile, die die Kundennummer des Entleihers und die Inventarnummer des entliehenen Buches angibt. Die folgende Tabelle

kunden_nr	inventar_nr
24	11
24	24
31	18
44	19

besagt, dass Frau Müller die Bücher „Introduction to Database Systems“ und „Einführung in die Volkswirtschaftslehre“ ausgeliehen hat. Sowohl Herr Huber als auch Frau Kiefer haben jeweils ein Exemplar von „ANSI Common Lisp“ entliehen, durch die Inventarnummer kann man genau feststellen welches Exemplar von Nicole Kiefer und welches von Hans Huber entliehen wurde.

Oben wurde vorausgesetzt, dass keine zwei Kunden mit der gleichen Kundennummer in der Datenbank existieren. Daher kann man einen Verleihvorgang immer höchstens einem Kunden zuordnen. Es wäre allerdings auch möglich, dass die Tabelle `verleih` Zeilen enthält, für die `kunden_nr` nicht in der Tabelle `kunden` vorkommt. Da derartige Einträge keinen Sinn ergeben, will man sie in der Datenbank nicht zulassen.

Formaler lässt sich die im vorhergehenden Absatz formulierte Bedingung so ausdrücken: Die Kundennummer ist ein Schlüsselkandidat in der Tabelle `kunden` und jede in der Tabelle `verleih` vorkommende Kundennummer muss auch in der Tabelle `kunden` vorkommen. Ebenso muss jede Buchnummer die Inventarnummer eines

tatsächlich in der Bibliothek vorhandenen Buches sein. Derartige Beziehungen werden durch den Begriff des Fremdschlüssels erfasst.

Definition 4.1. Ein Fremdschlüssel F einer Tabelle R bezüglich einer Tabelle S ist eine Menge von Spalten von R, so dass F ein Schlüsselkandidat von S ist. Das DBMS garantiert, dass eine Zeile aus R für alle in F enthaltenen Spalten nur Werte annehmen kann, die auch in S vorkommen.

Man kann also sagen, `kunden_nr` ist ein Fremdschlüssel der Tabelle `verleih` bezüglich der Tabelle `kunden`. Beachten Sie, dass die Spalte `kunden_nr` *kein* Schlüsselkandidat für die Tabelle `verleih` ist, ein Fremdschlüssel F einer Tabelle S muss also nicht Schlüsselkandidat (und insbesondere nicht Primärschlüssel) von S sein. Es gibt aber immer eine Tabelle, für die F Schlüsselkandidat ist.

Gewährleistet ein DBMS die Einhaltung der Bedingung für Fremdschlüssel, so sagt man auch es garantiert die *referentielle Integrität* der Datenbank.