Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Datenbanken

Vorlesungsskript für das 3. Semester

Studiengang Internationale Medieninformatik

4. Dreiwertige Logik und Primärschlüssel in

relationalen Datenbanksystemen

Dozent: M. Sc. Burak Boyaci

Version: 28.10.2025

Wintersemester 25/26

5

NULL-Werte und dreiwertige Logik

Kapitelübersicht

5.1 Unbekannte Information

In der Praxis haben wir es oft mit unvollständiger oder unbekannter Information zu tun. So kann es beim Erfassen von Personendaten vorkommen, dass wir das Geburtsdatum oder den Wohnort nicht kennen; bei einem Artikel wissen wir das Verkaufsdatum noch nicht, solange er noch auf Lager ist. Es gibt grundsätzlich verschiedene Möglichkeiten, in einer Datenbank mit fehlender Information umzugehen. Codd löste das Problem für relationale Datenbanken als Teil seiner „3. Regel“:

Definition 5.1. Fehlende Information eines Attributs wird durch den speziellen Wert **NULL** dargestellt, unabhängig vom Datentyp des Attributs.

Der Wert **NULL** ist damit ein *einheitlicher* Repräsentant für unbekannte Information.

Beispiel 5.2. Nehmen wir für das Beispiel 3.4 unserer Comic-Datenbank aus dem Datenbanken 2 Grundlagen SQL Skript auf Seite 12 an, wir hätten ein Album entdeckt, das wir eintragen möchten, ohne allerdings den Band und den Preis zu kennen. Mit der folgenden INSERT-Anweisung können wir diesen Eintrag in der Tabelle alben speichern:

```
INSERT INTO alben (titel, reihe, jahr) VALUES ('Lucky Luke' , 'Lucky Luke' ,1976) ;
```

Mit

```
SELECT * FROM alben ;
```

erhält man dann:

reihe	titel	band	preis	jahr
Asterix, der Gallier	Asterix	1	2.80	1968
Asterix und Kleopatra	Asterix	2	2.80	1968
Gespenster Geschichten	Gespenster Geschichten	1	1.20	1974
Die Trabantenstadt	Asterix	17	3.80	1974
Der große Graben	Asterix	25	5.00	1980
Das Kriminalmuseum	Franka	1	8.80	1985
Das Meisterwerk	Franka	2	8.80	1986
Lucky Luke	Lucky Luke	NULL	NULL	1976

SQL verwendet also tatsächlich für unbekannte Information den Wert **NULL**.

Will man explizit verhindern, dass ein Attribut **NULL** sein kann, so müssen wir es bei der Erzeugung der Tabelle mit der Einschränkung **NOT NULL** versehen, also zum Beispiel:

```
CREATE TABLE personen (
name varchar(20) NOT NULL,
alter smallint
) ;
```

Ein Eintragsversuch mit

```
INSERT INTO personen (alter) VALUES (24) ;
```

oder

```
INSERT INTO personen (name, alter) VALUES (NULL, 24) ;
```

führt dann jeweils zu einer Fehlermeldung. Die Tabelle ist damit so konstruiert, dass sie einen Eintrag ohne eine Wertangabe für die spezifizierten Attribute gar nicht erst ermöglicht.

Die Erstellung eines Tabellenattributs mit der Einschränkung **NOT NULL** ist eine vonmehreren Möglichkeiten in SQL, sogenannte *Integritätsregeln*[1] zu implementieren. Integritätsregeln spielen eine wichtige Rolle bei den Beziehungen von Datensätzen unterschiedlicher Tabelle. Wir werden darauf zu einem späteren Zeitpunkt noch zu sprechen kommen.

5.2 Dreiwertige Logik

Mit dem Wert **NULL** für unbekannte Information ist zwar ein Problem der realen Welt gelöst. Welchen Wahrheitswert jedoch soll eine Abfrage auf den Wert **NULL** ergeben? Ist die logische Aussage

```
'Otto' = NULL
```

wahr oder falsch? Um das herauszufinden, betrachten wir anhand des folgenden Beispiels, wie SQL hier entscheidet.

Beispiel 5.3. Betrachten wirzur Verdeutlichung das folgende vereinfachte Beispiel einer Tabelle *logik (aussage, wahrheitswert)* mit zwei Attributen *aussage* und *wahrheitswert*, die logische Aussagen und ihre jeweiligen Wahrheitswerte speichern sollen. Also beispielsweise die Aussage „1 + 2 = 3“ mit ihrem Wahrheitswert true. Die Tabellenstruktur wird dann mit der Anweisung

```
CREATE TABLE logik (aussage varchar(100), wahrheitswert boolean);
```

[1] Piepmeyer (2011):S. 88ff.

erzeugt. Speichern wir nun die Aussagen „1+ 2 = 3“ und „2 + 3 = 4“ mit ihren jeweiligen Wahrheitswerten:

```
INSERT INTO logik (aussage, wahrheitswert) VALUES
  ('1+2=3'  , true),
  ('2+3=4'  , false) ;
```

Wie speichern wir jedoch die Aussage „Ich werde eine 6 würfeln“? Wir wissen nicht, ob sie wahr oder falsch ist, also speichern wir sie ohne ihren Wahrheitswert:

```
INSERT INTO logik (aussage) VALUES ('Ich werde eine 6 würfeln') ;
```

Setzen wir die **SELECT**-Anweisungen

```
SELECT * FROM logik ;
SELECT * FROM logik WHERE wahrheitswert OR NOT wahrheitswert ;
SELECT * FROM logik WHERE (wahrheitswert =NULL) OR NOT (wahrheitswert = NULL) ;
```

darauf ab, so erhalten wir jeweils die folgenden Ergebnismengen:

```
SELECT * FROM logik;
```

aussage	wahrheitswert
1+2 = 3	TRUE
2+3 = 4	FALSE
Ich werde eine 6 würfeln	NULL

```
SELECT * FROM logik WHERE
wahrheitswert
OR
NOT wahrheitswert
```

aussage	wahrheitswert
1+2 = 3	TRUE
2+3 = 4	FALSE

```
SELECT * FROM logik WHERE
wahrheitswert = NULL
OR
NOT wahrheitswert = NULL
```

aussage	wahrheitswert

In der Boole’schen Algebra ist eine Aussage stets wahr oder falsch, d.h. die **OR**-Verknüpfung eines beliebigen Wahrheitswert mit seiner Verneinung, also

wahrheitswert **OR NOT** wahrheitswert

ist in der klassischen Logik immer wahr. Bei allen drei **SELECT**-Anweisungen dürften wir also erwarten, dass sie jeweils *alle* Datensätze der Tabelle anzeigen. Haben wir also einen schwerwiegenden Fehler in SQL gefunden? □

Tatsächlich sind die Abfrageergebnisse mit den WHERE-Klauseln aus dem obigen Beispiel nur konsequent, was wir mit der dritten Abfrage sofort herleiten können:

Regel 1. Jeder *Vergleich eines Wertes mit **NULL** ist weder wahr noch falsch, sondern unbekannt, also **NULL**.*

Warum ist das nur konsequent? Der Wert **NULL** ist weder gleich noch ungleich einem anderen Wert, insbesondere aber kann **NULL** nach Definition 5.1 als unbekannter Wert weder gleich noch ungleich sich selbst (eigentlich ja einem anderen unbekannten Wert) sein.

Bemerkung 5.4. Da **NULL** unbekannte Information darstellt, muss konsequenterweise auch jeder Vergleichmit **NULL** das Resultat „unbekannt“ ergeben. Wederwahr noch falsch sinddaher korrekt! Vergleiche in SQL basieren daher nicht auf der klassischen Logik mit zwei logischen Werten, sondern auf einer *drei*wertigen Logik mit den drei Wahrheitswerten *w* (wahr), *f* (falsch) und

u (unbekannt). Der polnische Mathematiker Jan Łukasiewicz hat eine solche Logik, heute genannt, bereits 1920 entwickelt.[2] Sie ist durch die folgenden Wahrheitstabellen gegeben:

x	**NOT** x
f	w
u	u
w	f

AND	f	u	w
f	f	f	f
u	f	u	u
w	f	u	w

OR	f	u	w
f	f	u	w
u	u	u	w
w	w	w	w

(5.1)

Die Verneinung **NOT** x eines Wahrheitswertes x ist hier in der ersten Tabelle zeilenweise durch die zweite Spalte angegeben, während die zweite und dritte Tabelle jeweils so zu lesen sind, dass die die beiden Argumente im Spalten- und Zeilenkopf stehen und ihre Verknüpfung im Innern der Tabelle. (Also beispielsweise u **AND** $w = u$.) □

Bemerkung * 5.5. *(Numerische Algebra der Wahrheitswerte)* Man kann mit Wahrheitswerten rechnen wie mit Zahlen. Mit den Entsprechungen 0 ↔falsch, $\frac{1}{2}$↔ unbekannt und 1 ↔ wahr:

x	**NOT** x
0	1
$\frac{1}{2}$	$\frac{1}{2}$
1	0

AND	0	$\frac{1}{2}$	1
0	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
1	0	$\frac{1}{2}$	1

OR	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

(5.2)

(Der Wahrheitswert u liegt hier also *zwischen* f und w.) Ein Vorteil dieser Darstellung der Wahrheitswerte ist, dass die logischen Operatoren als numerische Verknüpfungen aufgefasst werden können:

$$\textbf{NOT}\ a = 1 - a, \qquad a\ \textbf{AND}\ b = \min(a, b), \qquad a\ \textbf{OR}\ b = \max(a, b) \tag{5.3}$$

Diese Version der Łukasiewicz'schen Logik L_3 ist nicht unüblich.[3]

Zwischenfrage 5.6. Welche Ergebnismengen liefern die folgenden Anweisungen für die Tabelle logik aus Beispiel 5.3?

```
SELECT wahrheitswert FROM logik WHERE (1 = 1) OR (NULL = NULL);
SELECT wahrheitswert FROM logik WHERE (1 = 1) AND (NULL = NULL);
```

5.3 Abfragen auf NULL-Werte

Wie können wir nun herausfinden, ob ein Attribut eines Datensatzes den Wert **NULL** hat? Mit dem Gleichheitszeichen =geht es ja nun nicht. Für diesen Zweck können wir aber das reservierte Wort **IS** verwenden. Für den umgekehrten Fall steht uns **IS NOT** zur Verfügung. Beispielsweise ergeben die folgenden Anfragen für unser Beispiel 5.3:

```
SELECT aussage FROM logik
WHERE wahrheitswert IS NULL;
```

aussage
Ich werde eine 6 würfeln

```
SELECT aussage FROM logik
WHERE wahrheitswert IS NOT NULL;
```

aussage
1+2 = 3
2+3 = 4

[2] vgl. de Vries (2007):§4.2.
[3] de Vries (2007):§4.2.

5.4 Setzen von Standardwerten bei NULL-Einträgen

Oft benötigt man Datenauswertungen, in denen ein NULL-Wert einen berechenbaren Wert, z.B. 0, erhält. Dazu gibt es die Funktion

IF NULL(*x*, neu),

die einen NULL-Wert durch den Wert neu ersetzt, aber ansonsten den originalen Wert *x* zurückgibt. Bei MS Access heißt diese Funktion **Nz**, bei Oracle **NVL**.

Beispiel 4.7. Wollen wir uns in der Ausgabe unserer Logiktabelle in Bemerkung 4.5 für unbekannte Wahrheitswerte den Wert $\frac{1}{2}$ anzeigen, so erhalten wir

```
SELECT aussage, IF NULL(wahrheitswert, 0.5) AS wahrheitswert FROM logik;
```

die Ergebnismenge

aussage	wahrheitswert
1 + 2 = 3	1
2 + 3 = 4	0
Ich werde eine 6 würfeln	0,5

wie in der Łukasiewicz-Logik.

6

Primärschlüssel: Identifikation von Datensätzen

Kapitelübersicht

6.1 Schlüssel

Ein wichtiges Merkmal einer relationalen Datenbank ist, dass sie keine völlig identischen Datensätze enthalten darf. Jeder Datensatz muss also irgendeine Information enthalten, die sonst kein anderer Datensatz der Tabelle hat. Um dies für eine Datenbank technisch zu gewährleisten, gibt es den Begriff des „Schlüssels“: Ein *Schlüssel* ist eine Kombination von Attributen einer Tabelle, die *eindeutig* ist in dem Sinne, dass sie für jeden Datensatz eine andere Kombination von Attributwerten aufweist.

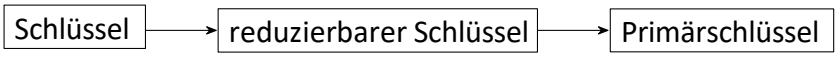
Im Extremfall ist nur die Kombination aller Attributwerte eindeutig, d.h. der Schlüssel muss dann aus allen Attributen zusammengesetzt sein. Durch seinen Schlüssel ist also jeder Datensatz einer Tabelle eindeutig identifizierbar. In unserer „Full House“-Tabelle aus Beispiel 2.1 wäre z.B. die Kombination (farbe, karte) ein Schlüssel, nicht aber eines der Attribute farbe oder karte allein:

farbe	karte
Kreuz	Ass
Pik	Ass
Herz	König
Herz	Ass
Karo	König

Ein Schlüssel ist *reduzierbar*, falls er für die Tabelle eindeutig bleibt, auch wenn ein Feld aus ihm entfernt wird. Z.B. ist die Kombination (reihe, titel) unserer Comicalben-Tabelle reduzierbar, denn das Attribut reihe darf weggelassen werden, ohne dass die Eindeutigkeit verloren geht. Damit kann der für relationale Datenbanken wichtigste Schlüssel, der sogenannte Primärschlüssel, definiert werden:

Definition 6.1. Der *Primärschlüssel* einer Tabelle ist ein nicht reduzierbarer Schlüssel, der zur Identifikation der Datensätze ausgewählt wurde. Er kann in der Praxis nachträglich meist nur schwer geändert werden.

Die Beziehung der verschiedenen Schlüsselbegriffe wird durch die folgende Grafik illustriert:



Es gibt zwei Möglichkeiten in SQL, den Primärschlüssel einer Tabelle schon bei ihrer Erzeugung bestimmen, zum Einen mit der Anweisung:

```
CREATE TABLE tabelle (
    spalte_1  datentyp_1,
    spalte_2 datentyp_2,
       ...
    spalte_n datentyp_n,
    PRIMARY KEY (spalte_x, ..., spalte_y)
);
```

zum anderen mit der Anweisung

```
CREATE TABLE tabelle (
    spalte_1 datentyp_1 PRIMARY KEY,
    spalte_2 datentyp_2,
       ...
    spalte_n datentyp_n
);
```

In der ersten Variante bestimmt die in den Klammern nach **PRIMARY KEY** aufgeführte Spaltenauswahl diejenigen Spalten der Tabelle, die ihren Primärschlüssel bilden. In der zweiten Variante ist die Spalte, in der der Ausdruck **PRIMARY KEY** steht, gleichzeitig ihr Primärschlüssel. Ein zusammengesetzter Primärschlüssel muss über die erste Variante abgebildet werden.

Bei der Darstellung des Typs der Tabelle in der Notation (1.5) für den Relationstyp werden die Attribute des Primärschlüssels unterstrichen:

tabelle (spalte_x, ..., spalte_y, spalte_z, ...)

6.2 Wahl eines natürlichen Primärschlüssels

Bildet man den Primärschlüssel aus einer Attributkombination (oder einem einzelnen Attribut) einer Tabelle, so spricht man auch von einem „natürlichen“ Primärschlüssel. Obwohl die Eigenschaften eines Primärschlüssels wohldefiniert sind, ist die Festlegung des Primärschlüssels einer gegebenen Tabelle nicht immer einfach und hängt von dem konkreten Anwendungsfall und seinem Kontext ab.

Beispiel 6.2. Betrachten wir ein einfaches Beispiel, eine Tabelle namens artikel. Sie soll alle Artikel eines Werkzeuge produzierenden Betriebes speichern, als Grundlage für einen Angebotskatalog. Ein Artikel ist spezifiziert durch seine Bezeichnung, seinen Preis und seine Farbe:

artikel

bezeichnung	preis	farbe
Hammer	15 €	braun
Hammer	15,20 €	rot
Mottek	45,10 €	schwarz
Säge	25 €	blau
Säge	24,90 €	grün
Zange	5,49 €	rot

Die Spalten sind die Felder der Tabelle, und jede Zeile ein Datensatz, also ein spezieller Artikel. In unserer Artikeltabelle ist z.B. die Kombination (*bezeichnung, preis, farbe*)ein Schlüssel, denn diese Kombination ist bei allen Artikeln verschieden. Dieser Schlüssel ist aber reduzibel, der Preis allein ist schon ein Schlüssel. Der Preis kann also in unserem Beispiel als ein Primärschlüssel unserer Tabelle gewählt werden — allerdings ein schlechter! Denn wer garantiert, dass nicht morgen ein Akkuschrauber für 15,20 € angeboten wird, und der Preis ist plötzlich kein Primärschlüssel mehr? Oder was passiert, wenn der Mottek plötzlich in einer Werbungsaktion den Sonderpreis von 30 € erhält? Ändert er dann seine „Identität“?

Ein Primärschlüssel sollte zeitlich stabil sein, d.h. seine Werte sollten sich während des Lebenszyklus der Tabelle nicht ändern. Da in der Praxis häufig Datensätze neu in Tabellen eingefügt werden, muss ein Primärschlüssel nicht nur für den aktuellen Datenbestand, sondern auch langfristig eindeutig bleiben. Einen Primärschlüssel zu finden kann also durchaus schwierig werden!

Beispiel 6.3. Unsere Comicsammlung aus Beispiel 3.4 haben wir ohne einen Primärschlüssel angelegt. Was wäre ein sinnvoller Schlüssel? Als erster Kandidat käme da vielleicht der Titel in Frage. Er ist auf jeden Fall stabil, denn er wird sich für ein Album nicht ändern. Aber ist er sicher immer eindeutig? Eigentlich nicht, denn vielleicht erscheint morgen ein Asterix-Album „Das Meisterwerk“, und das könnten wir dann nicht mehr speichern. Dagegen ist die Kombination (Reihe, Band) sicher eindeutig, also auch in Zukunft, denn die Verlage nummerieren ihre Reihen ja systematisch durch. Unser Relationstyp wäre damit also

alben (titel, reihe, band, preis, jahr),

d.h. in SQL:

```
CREATE TABLE alben (
  titel varchar(50),
  reihe varchar(50),
  band smallint,
  preis decimal(4,2),
  jahr smallint,
  PRIMARY KEY (reihe, band)
) ;
```

□

6.3 Primärschlüssel und NULL-Werte

Eine Besonderheit ergibt sich aus Regel 1 für Attribute eines Primärschlüssels. Denn für die Eindeutigkeit muss der Vergleich der Werte zweier Primärschlüssel entweder wahr oder falsch sein. Das hat die folgende Regel zur „Entitätsintegrität“ zur Konsequenz:

Definition 6.4. *(Entitätsintegrität)* Für ein Attribut eines Primärschlüssels ist der Wert **NULL** nicht zulässig. □

Schon bei der Entwicklung des Konzepts der relationalen Datenbanken schränkte Codd die Möglichkeiten von Attributwerten mit unbekannter Information durch seine „3. Regel“ ein. Entsprechend wird die Entitätsintegrität von SQL (in allen Dialekten) unterstützt. Geben wir in Beispiel 5.3 beispielsweise die Anweisung

```
INSERT INTO alben (titel, reihe, jahr) VALUES ('Lucky Luke' , 'Lucky Luke' ,1976);
```

ein, so erhalten wir eine Fehlermeldung, denn das Attribut reihe darf als Bestandteil des Primärschlüssels nicht **NULL** sein. (Vgl. im Gegensatz dazu Beispiel 4.2 der Albentabelle ohne Primärschlüssel.)

6.4 Künstliche Primärschlüssel

Wegen der oben in Abschnitt 5.2 genannten Schwierigkeiten der Wahl eines „natürlichen“ Schlüssels aus der Kombination bestehender Attribute wird häufig ein *künstlicher Primärschlüssel (surrogate key)* als neues Attribut gewählt, der unabhängig von den realen Daten wie (Bezeichnung, Preis, Farbe) jeden Datensatz eindeutig identifiziert. Solch ein Feld ist zweckmäßigerweise eine eindeutige Nummer, oft kurz ID genannt. Das ist der Grund, warum Sie in jedem Verein eine Mitgliedsnummer, in der Hochschule eine Matrikelnummer oder bei einer Firma eine Kundennummer haben! SQL sieht sogar den eigenen Datentyp **SERIAL**vor, mit dem eine solche ID automatisch verwaltet wird, indem sie bei jedem neu einzufügenden Datensatz hochgezählt wird:

```
CREATE TABLE kanten (
  id SERIAL PRIMARY KEY,
  ...
);
```

oder

```
CREATE TABLE kanten
  ( id SERIAL,
  ...
  PRIMARY KEY (id)
);
```

Die Anweisung bestimmt also das spezifizierte Attribut als ganzzahligen und automatisch verwalteten Primärschlüssel. Bei einigen RDBMS wie MS Access heißt dieser Datentyp auch *autoincrement*. Ein solcher künstlicher Primärschlüssel heißt entsprechend *seriell*. Wir dürfen beim Einfügen von Datensätzen in eine Tabelle mit einem seriellen Primärschlüssel dessen Wert jedoch nicht festlegen, sondern müssen ihn einfach weglassen.

Beispiel 6.5. Legen wir unsere Comicsammlung aus Beispiel 3.4 und 4.2 mit einem seriellen Primärschlüssel an:

```
CREATE TABLE alben (
  id serial primary key,
  titel varchar(50),
  reihe varchar(50),
  band smallint,
  preis decimal(4,2),
  jahr smallint
) ;
```

so können wir unsere Alben mit den Anweisungen

```
INSERT INTO alben (titel, reihe, band, preis, jahr) VALUES
  ('Asterix, der Gallier'      , 'Asterix', 1, 2.80, 1968),
  ('Asterix und Kleopatra'   , 'Asterix', 2, 2.80, 1968),
  ('Gespenster Geschichten' , 'Gespenster Geschichten' , 1, 1.20, 1974),
  ('Die Trabantenstadt' , 'Asterix', 17, 3.80, 1974),
  ('Der große Graben' , 'Asterix', 25, 5.00, 1980),
  ('Das Kriminalmuseum' , 'Franka', 1, 8.80,1985),
  ('Das Meisterwerk' , 'Franka', 2, 8.80, 1986) ;
```

und

```
INSERT INTO alben (titel, reihe, jahr) VALUES ('Lucky Luke' , 'Lucky Luke' , 1976) ;
```

speichern. Insbesondere können wir also das Album Lucky Luke trotz unbekannter Bandnummer problemlos eintragen. Betrachten wir die gespeicherten Daten mit

```
SELECT * FROM alben;
```

so erhalten wir die Ergebnismenge

id	titel	reihe	band	preis	jahr
1	Asterix, der Gallier	Asterix	1	2.80	1968
2	Asterix und Kleopatra	Asterix	2	2.80	1968
3	Gespenster Geschichten	Gespenster Geschichten	1	1.20	1974
4	Die Trabantenstadt	Asterix	17	3.80	1974
5	Der große Graben	Asterix	25	5.00	1980
6	Das Kriminalmuseum	Franka	1	8.80	1985
7	Das Meisterwerk	Franka	2	8.80	1986
8	Lucky Luke	Lucky Luke	null	null	1976

Wir sehen, dass das RDBMS den seriellen Primärschlüssel automatischhochgezählt hat.