Einführung in Datenbanken

Kapitel 9: Nullwerte in SQL

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2019/20

http://www.informatik.uni-halle.de/~brass/db19/

Inhalt

Beispiel-Datenbank

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

Nullwerte (1)

- Nullwerte werden in vielen verschiedenen Situationen verwendet, z.B.:
  - Ein Wert existiert, ist aber unbekannt.

    Angenommen, man will in der Tabelle `STUDENTEN` auch die Telefonnummer der Studenten speichern, aber man kennt möglicherweise nicht von jedem Studenten die Telefonnummer, obwohl wahrscheinlich alle irgendwie telefonisch zu erreichen sind.
  - Es existiert kein Wert.

    In einer Tabelle mit Vorlesungsdaten könnte es eine Spalte `URL` geben, aber nicht jede Vorlesung hat eine Web-Seite (wenn aber eine Webseite existiert, wäre sie normalerweise auch eingetragen).
  - Es könnte ein (unbekannter) Wert existieren, oder auch keiner.

Nullwerte (2)

- Anwendungen von Nullwerten, fortgesetzt:
  - Das Attribut ist auf dieses Tupel nicht anwendbar.

    Z.B. müssen an einer Universität in den USA nur ausländische Studenten einen Toefl-Test ablegen, um ihre Englischkenntnisse zu beweisen.
    Eine Spalte für die Toefl-Punktzahl in der Tabelle `STUDENTEN` ist für U.S.-Studenten nicht anwendbar. Selbst wenn diese Studenten früher einmal einen Toefl-Test gemacht haben (z.B. weil sie Immigranten sind), ist die Universität an dem Resultat nicht interessiert.
  - Wert wird später zugewiesen/bekannt gegeben.
  - Jeder Wert ist möglich.
- Ein Ausschuss fand 13 verschiedene Bedeutungen von Nullwerten.

Nullwerte (3)

Vorteile von Nullwerten:

- Ohne Nullwerte wäre es nötig, die meisten Relationen in viele aufzuspalten ("Subklassen"):
  - Z.B. `STUDENTEN_MIT_EMAIL`, `STUDENTEN_OHNE_EMAIL`.
  - Oder extra Relation: `STUD_EMAIL(SID, EMAIL)`.
  - Das erschwert Anfragen.

    Man braucht Verbunde und Vereinigungen (siehe Kapitel 10).
- Sind Nullwerte nicht erlaubt, werden sich die Nutzer Werte ausdenken, um die Spalten zu füllen.

  Das macht die DB-Struktur sogar noch unklarer und führt ggf. zu falschen Resultaten bei statistischen Auswertungen.

Nullwerte (4)

Probleme:

- Da der gleiche Nullwert für verschiedene Zwecke genutzt wird, kann es keine klare Semantik geben.
- SQL benutzt dreiwertige Logik, um Bedingungen mit Nullwerten auszuwerten.

  Da man an die normale zweiwertige Logik gewöhnt ist, kann es Überraschungen geben — einige Äquivalenzen gelten nicht.
- Fast alle Programmiersprachen haben keine Nullwerte. Das erschwert Anwendungsprogramme.

  Wenn also ein Attributwert in eine Programmvariable eingelesen wird, muss er auf Nullwerte überprüft werden ($\rightarrow$ Indikatorvariablen).

Inhalt

Terme mit Nullwerten (1)

- Tupel/Tabellenzeilen können einen Nullwert enthalten (falls dies nicht mit `NOT NULL` für das Attribut/die Spalte in der Tabellendeklaration ausgeschlossen wurde).
- Der Nullwert ist von allen normalen Werten des Datentyps verschieden, insbesondere ist er verschieden von der Zahl 0 und dem leeren String.

  Oracle identifiziert den Nullwert und den leeren String. Das ist eine Verletzung des SQL-Standards (Oracle listet dies als "nicht-übereinstimmend" in einem Anhang seiner SQL-Referenz auf). Da es in Anwendungsprogrammen verwendet werden könnte und es existierende DB-Dateien gibt, die nicht zwischen beiden unterscheiden, ist es schwierig zu ändern.

- Ist eines der Argumente Null, so liefern Datentyp-Funktionen auch Null. Ist z.B. `A` Null, so ist `A+B` ebenfalls Null.

Terme mit Nullwerten (2)

- Das Schlüsselwort `NULL` ist selbst kein Term (Ausdruck), obwohl es an vielen Stellen verwendet werden kann, an denen ein Term verlangt wird.
- `NULL` hat keinen Datentyp, also braucht man einen Kontext, so dass der Typ klar ist:
  - In SQL-92 und DB2 liefert `CAST(NULL AS type)` einen Nullwert des angegebenen Typs.
  - In Oracle kann `NULL` oft als Term verwendet werden, aber z.B. dies ist ein Fehler:
    ```
    SELECT 1 FROM DUAL UNION SELECT NULL FROM DUAL
    ```
    Man muss `TO_NUMBER(NULL)` schreiben, sonst unterschiedliche Typen.
  - In SQL Server, Access, MySQL wird "`NULL`" als normaler Term verwendet (mit beliebigem Typ).

Inhalt

Dreiwertige Logik (1)

- Betrachten Sie folgende Anfrage:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  EMAIL = 'xyz@acm.org'
```

- Was passiert, wenn ein Student in der Spalte `EMAIL` einen Nullwert hat? Er wird nicht ausgegeben.

- Aber er tritt auch nicht im Ergebnis dieser Anfrage auf (weil der Wert nicht bekannt ist):

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  NOT (EMAIL = 'xyz@acm.org')
```

Dreiwertige Logik (2)

- Die Bedingung

  ```
  EMAIL = 'xyz@acm.org'
  ```

  ist nicht falsch, wenn `EMAIL` Null ist, da sonst die Zeile in der negierten Anfrage auftauchen würde.

  Natürlich ist sie auch nicht wahr.

- SQL verwendet eine dreiwertige Logik, um Nullwerte zu behandeln. Die drei Wahrheitswerte sind `wahr`, `falsch` und `unbekannt`.

  Anstelle von "`unbekannt`" liest man auch oft "`Null`".

- Die Idee ist, dass Tupel "herausgefiltert" werden sollten, die einen Nullwert in einem Attribut haben, welches für die Anfrage wichtig ist — sie sollten das Anfrageergebnis nicht beeinflussen.

Dreiwertige Logik (3)

P	Q	NOT P	P AND Q	P OR Q
falsch	falsch	wahr	falsch	falsch
falsch	unbekannt	wahr	falsch	unbekannt
falsch	wahr	wahr	falsch	wahr
unbekannt	falsch	unbekannt	falsch	unbekannt
unbekannt	unbekannt	unbekannt	unbekannt	unbekannt
unbekannt	wahr	unbekannt	unbekannt	wahr
wahr	falsch	falsch	falsch	wahr
wahr	unbekannt	falsch	unbekannt	wahr
wahr	wahr	falsch	wahr	wahr

Man braucht diese Tabelle nicht auswendig zu lernen. Man setze einfach anstelle von “unbekannt” probehalber “wahr” und “falsch”ein. Wenn es unterschiedliche Ergebnisse gibt, ist das Resultat “unbekannt”, sonst das eindeutige Ergebnis. D.h.: `AND`, `OR`, `NOT` leiten den Wahrheitswert “unbekannt” weiter, außer das Ergebnis ist klar.

Dreiwertige Logik (4)

- In SQL ergibt ein Vergleich mit einem Nullwert immer den dritten Wahrheitswert "`unbekannt`".

  Der wahre Attribut-Wert ist unbekannt oder existiert nicht, also wäre es falsch zu sagen, dass das Ergebnis eines Vergleichs mit einem Nullwert wahr oder falsch ist.

- Eine Ergebniszeile wird nur dann ausgegeben, wenn die `WHERE`-Bedingung "`wahr`" ist.

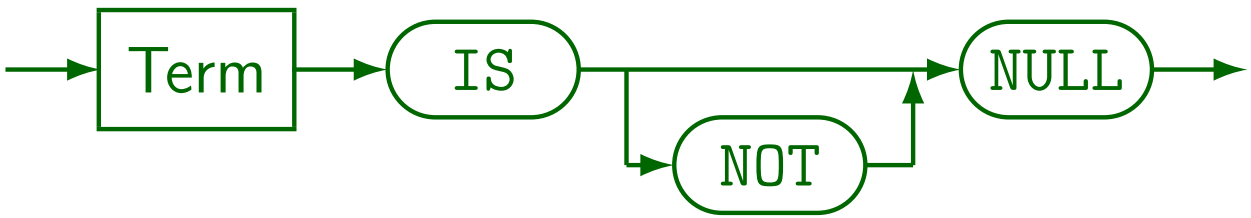
- Somit hat folgende Anfrage ein leeres Ergebnis:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  EMAIL = null
```

Die Anfrage ist eigentlich illegal in SQL-92, DB2 lehnt sie ab. Oracle, SQL Server, Access, MySQL akzeptieren sie und geben das leere Ergebnis aus.

Test auf Null (1)

Bedingung (Form 8: IS NULL):



- Beispiel: `EMAIL IS NULL`
- Man beachte, dass `EMAIL = NULL` nicht funktioniert!

  Siehe vorige Folie. In Oracle und SQL Server ist dies immer "unbekannt" (nicht "wahr" oder "falsch") und in SQL-92 und DB2 ist es ein Syntax-Fehler. In SQL Server 7 funktioniert "`EMAIL = NULL`" nach dem Befehl "`SET ANSI_NULLS OFF`" (dann wird zweiwertige Logik verwendet).

- `EMAIL NOT NULL` ist ein Syntax-Fehler ("`IS`" fehlt).

Test auf Null (2)

- Übung: folgende Anfrage gibt alle Studenten mit einer Email-Adresse in der Domäne ".pitt.edu" aus:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN
WHERE  EMAIL IS NOT NULL
AND    EMAIL LIKE '%.pitt.edu'
```

  Ist der Test auf Null notwendig?

- CHECK-Integritätsbedingungen sind erfüllt, wenn die Bedingung den Wert "unbekannt" hat.

  Sie sind nur verletzt, wenn die Bedingung falsch ist.

Probleme mit Nullwerten (1)

- Für diejenigen, die an die normale zweiwertige Logik gewöhnt sind (alle von uns), können Nullwerte manchmal zu Überraschungen führen: Manche logischen Äquivalenzen gelten in SQL nicht.
- Zählt man z.B. alle Studenten mit einer Email-Adresse in der Domäne "`.pitt.edu`" und alle Studenten mit einer anderen Email-Adresse, vermutet man normalerweise, alle Studenten zu erhalten.
- Das ist in SQL nicht wahr — diejenigen mit einem Nullwert in der `EMAIL`-Spalte werden nicht gezählt.

Probleme mit Nullwerten (2)

- Z.B. ist `X = X` “unbekannt” und nicht “wahr”, wenn `X` Null ist.
- Da ein Nullwert verschiedene Bedeutungen haben kann, kann es keine zufriedenstellende Semantik für eine Anfragesprache geben.

  Z.B. würde die Bedeutung “Wert existiert, ist jedoch unbekannt” ($\exists X\colon \ldots$) die Verwendung der normalen logischen Äquivalenzen erlauben.

Inhalt

IN vs. EXISTS (1)

- `IN`-Bedingungen sind praktisch, aber nicht wirklich nötig: Man kann jede `IN`-Bedingung in eine äquivalente `EXISTS`-Bedingung übersetzen.
- Die Bedingung

$$t_1 \texttt{ IN (SELECT } t_2 \texttt{ FROM } R_1\, X_1, \ \ldots, \ R_n\, X_n \texttt{ WHERE } \varphi\texttt{)}$$

  ist (unter gewissen Voraussetzungen) äquivalent zu

$$\texttt{EXISTS (SELECT * FROM } R_1\, X_1, \ \ldots, \ R_n X_n \texttt{ WHERE (}\varphi\texttt{) AND } t_1 = t_2\texttt{)}$$

IN vs. EXISTS (2)

- Voraussetzung ist, dass die Bedeutung von t_1 nicht verändert wird, wenn es in die Unteranfrage verschoben wird:
  - Alle Tupelvariablen, die in t_1 vorkommen, müssen verschieden von $X_1, \ldots, X_n$ sein (ggf. X_i umbenennen).
  - Enthält t_1 Attributreferenzen A ohne Tupelvariable, so dürfen die R_i kein Attribut A haben (ggf. Tupelvariable einfügen).
- Außerdem gilt die Äquivalenz nur, wenn die Unteranfrage für t_2 keine Nullwerte liefert. Falls die Unteranfrage nur Werte liefert, die verschieden von t_1 sind, und einen Nullwert, so
  - Hat die `IN`-Bedingung den dritten Wahrheitswert "unbekannt".

    Sie wird wie eine große Disjunktion von Gleichungen $t_1 = c$ behandelt, wobei für c alle Werte eingesetzt werden, die die Unteranfrage liefert.
  - Die `EXISTS`-Bedingung ist dagegen "falsch".

IN vs. EXISTS (3)

- Der Unterschied zwischen den Wahrheitswerten "unbekannt" und "falsch" ist wichtig, wenn anschließend eine Negation erfolgt (wegen `NOT IN`).
- Beispiel: Die Punkte-DB sei um eine Tabelle mit Kapiteln der Vorlesung erweitert. `THEMA` in `AUFGABEN` verweist jetzt auf diese Tabelle und kann auch Null sein:

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	NULL	14

KAPITEL

THEMA	...
Einführung	...
ER	...
SQL	...

IN vs. EXISTS (4)

- Die Anfrage nach Kapiteln ohne Aufgaben funktioniert nicht, wenn sie mit `IN` formuliert wird:

```
SELECT K.THEMA
FROM   KAPITEL K
WHERE  K.THEMA NOT IN (SELECT A.THEMA
                       FROM   AUFGABEN A)
```

- Die Ausgabe ist leer, obwohl intuitiv "`Einführung`" herauskommen sollte.
- Grund ist der Nullwert, den die Unteranfrage liefert.

  Vermutlich ist es fast immer ein Fehler, "`NOT IN`" mit einer Unteranfrage zu verwenden, die Nullwerte liefern kann.

IN vs. EXISTS (5)

- Die entsprechende Anfrage mit `EXISTS` funtioniert dagegen (es wird "`Einführung`" ausgegeben):

```
SELECT K.THEMA
FROM   KAPITEL K
WHERE  NOT EXISTS(SELECT *
                  FROM   AUFGABEN A
                  WHERE  A.THEMA = K.THEMA)
```

- Dies zeigt wieder die Tücken dreiwertiger Logik.

  Bei der `NOT EXISTS`-Bedingung entsteht der "unbekannt" im Innern der Unteranfrage. Die Unteranfrage behandelt ihn dann wie "falsch" (sie gibt ja in diesem Fall nichts aus). Bei "`NOT IN`" entsteht der dritte Wahrheitswert erst außerhalb der Unteranfrage. Bei "`NOT IN`" muss man "`WHERE A.THEMA IS NOT NULL`" in der Unteranfrage fordern.

IN vs. EXISTS (6)

- Theoretisch ist interessant, dass man

t_1 `IN (SELECT` t_2
`FROM` $R_1\ X_1$`,` `...,` $R_n\ X_n$
`WHERE` φ`)`

exakt in eine `EXISTS`-Bedingung übersetzen kann:

```
    EXISTS (SELECT *
            FROM R1 X1, ..., RnXn
            WHERE (φ) AND t1 = t2)
OR 1 = NULL AND  -- Wahrheitswert "unbekannt"
    EXISTS (SELECT *
            FROM R1 X1, ..., RnXn
            WHERE (φ) AND t2 IS NULL)
```

Inhalt

COALESCE

- Die Funktion `COALESCE` kann verwendet werden, um einen Nullwert durch einen anderen Wert zu ersetzen.

  Mögliche Übersetzungen: “verschmelzen”, “sich verbinden”.

  `COALESCE` ist Teil des SQL-92 Standards. und ist verfügbar u.a. in u.a. in Oracle (seit Ver. 9i), PostgreSQL, DB2, MySQL, Microsoft SQL Server.

  Oracle hat alternativ die Funktion `NVL` mit zwei Argumenten, MySQL `IFNULL`.

- `COALESCE(`t_1`, `t_2`, ...)` liefert das erste Argument t_i, das nicht `NULL` ist.

  Sind alle Argumente `NULL`, ist das Ergebnis `NULL`.

- Beispiel: EMail-Adresse aller Studenten kann `NULL` sein:

```
SELECT NACHNAME, VORNAME,
       COALESCE(EMAIL, '(keine)') AS EMAIL
FROM   STUDENTEN
```

NULLIF

- Die Funktion `NULLIF` wandelt einen bestimmten Datenwert in einen Nullwert um.

  `NULLIF` ist Teil des SQL-92 Standards und wird unterstützt u.a. in Oracle (ab 9i), PostgreSQL, MySQL, DB2, Microsoft SQL Server.

- `NULLIF(`t_1`,` t_2`)` liefert `NULL`, falls $t_1 = t_2$, sonst t_1.

- Wenn man eine nicht existierende EMAIL-Adresse als "`(keine)`" abgespeichert hat, kann man dies so in den Nullwert übersetzen:

```
SELECT NACHNAME, VORNAME,
       NULLIF(EMAIL, '(keine)') AS EMAIL
FROM   STUDENTEN
```

  Im Ergebnis erscheint jetzt nicht "`(keine)`", sondern ein Nullwert.

Bedingte Ausdrücke (1)

- `COALESCE` und `NULLIF` können als Abkürzungen für bedingte Ausdrücke verstanden werden.
- Der SQL-92 Standard bietet bedingte Ausdrücke der folgenden Form:

```
CASE WHEN F1 THEN t1
     WHEN F2 THEN t2
     ...
     ELSE         tn+1
END
```

  Die `ELSE`-Klausel darf fehlen, dann wird implizit `ELSE NULL` angenommen.

- Dies ist ein Term (Wertausdruck). Er liefert den Wert des ersten Terms t_i, für den die Bedingung F_i wahr ist.

  Bedingte Ausdrücke gibt es z.B. auch in Sprachen wie C und Java: $(F)?t_1 : t_2$.

Bedingte Ausdrücke (2)

- COALESCE(t_1, t_2) entspricht:

  CASE WHEN t_1 IS NOT NULL THEN t_1 ELSE t_2 END

- Beispiel: Umrechnung der Buchstaben-Codes für die Aufgabentypen in ausführliche Namen:

```
SELECT CASE WHEN ATYP='H' THEN 'Hausaufgabe'
            WHEN ATYP='Z' THEN 'Zwischenklausur'
            WHEN ATYP='E' THEN 'Endklausur'
            ELSE 'Unbekannte Kat.' END,
       ANR, PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    VORNAME = 'Lisa' AND NACHNAME = 'Weiss'
```

Bedingte Ausdrücke (3)

- Der SQL-92 Standard erlaubt auch folgende Abkürzung:

```
SELECT CASE ATYP WHEN 'H' THEN 'Hausaufgabe'
                 WHEN 'Z' THEN 'Zwischenklausur'
                 WHEN 'E' THEN 'Endklausur'
                 ELSE 'Unbekannte Kat.' END,
       ANR, PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    VORNAME = 'Lisa' AND NACHNAME = 'Weiss'
```

In klassischen Oracle-SQL würde man die `DECODE`-Funktion benutzen:

```
DECODE(ATYP,'H','Hausaufgabe','Z','...','E','...','Unbekannte Kat.')
```