Einführung in Datenbanken

Kapitel 7: SQL: Datentypen

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

http://www.informatik.uni-halle.de/~brass/db20/

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Die möglichen Werte für die Datentypen `CHAR(`n`)`, `VARCHAR(`n`)`, `NUMERIC(`p`)` und `NUMERIC(`p`,`s`)` nennen.

  Und einige Kategorien weiterer Datentypen aufzählen.

- Einen passenden Datentyp für eine Tabellenspalte wählen.
- Die durch `+`, `-`, `*`, `/` und `||` bezeichneten Funktionen kennen und in Anfragen verwenden.

  Und Aufgaben, die weitere Datentyp-Funktionen benötigen, lösen (ggf. unter zielgerichteter Zuhilfenahme der DBMS-Dokumentation).

- Einen „case-insensitiven“ Vergleich mit `UPPER` ausführen.
- Den Unterschied zwischen der „pad space“ und „nopad space“ Vergleichsemantik für Zeichenketten erläutern.

Inhalt

Datentypen (1)

- Jede Spalte kann nur Werte eines bestimmten Datentyps speichern (unter `CREATE TABLE` definiert).
- Das relationale Modell hängt nicht von einer bestimmten Auswahl an Datentypen ab.
- Verschiedene DBMS bieten unterschiedliche Datentypen, aber Strings und Zahlen verschiedener Länge und Genauigkeit sind immer verfügbar.
- Moderne (objektrelationale) Systeme bieten auch benutzer-definierte Datentypen (Erweiterbarkeit).

  PostgreSQL, DB2, Oracle und SQL Server unterstützen benutzer-definierte Typen.

Datentypen (2)

- Datentypen definieren neben der Menge der möglichen Werte auch die Operationen auf den Werten.
- Im SQL-86-Standard gab es nur die Datentyp-Funktionen `+`, `-`, `*`, `/`.

  Die Existenz dieser Funktionen kann man in jedem DBMS erwarten.
- Andere Funktionen können sich von DBMS zu DBMS unterscheiden.

  Die Verwendung kann also zu Portabilitätsproblemen führen. Z.B. ist der Stringkonkatenations-Operator `||` im SQL-92-Standard enthalten, aber SQL Server und Access verwenden stattdessen „`+`“ und in MySQL heißt es „`concat(...)`“.

Datentypen (3)

Kategorien von Datentypen:

- Relativ standardisiert:
  - Zeichenketten (feste Länge, variable Länge)
  - Zahlen (Integer, Fest- und Gleitkommazahl)
- Unterstützt, aber in jedem DBMS verschieden:
  - Datums- und Zeitwerte
  - Lange Zeichenketten
  - Binäre Daten
  - Zeichenketten in nationalem Zeichensatz
- Benutzer-definierte und DBMS-spezifische Typen.

Datentypen in SQL ausprobieren

- Sie können ausprobieren, ob Ihr DBMS einen Typ T versteht, indem Sie ihn in einer Tabellendeklaration verwenden:

```
CREATE TABLE TEST(A T);
```

- Sie können dann auch ausprobieren, ob eine Konstante c einen Wert hat, der für den Typ akzeptabel ist:

```
INSERT INTO TEST VALUES(c);
```

- Zu Sicherheit sollten Sie sich die Tabelle nochmal anzeigen lassen, denn eventuell wurde der Wert bei der Einfügung umgewandelt/verändert:

```
SELECT * FROM TEST;
```

Z.B. gibt MySQL öfters keine Fehlermeldung aus, obwohl der Wert bei der Einfügung völlig verändert wird.

Datentyp-Funktionen in SQL ausprobieren

- Viele DBMS erlauben `SELECT` ohne `FROM`, dann können Sie Datentyp-Funktionen z.B. so ausprobieren:

```
SELECT UPPER('abc');
```

- Sie können sich aber auch eine Dummy-Tabelle machen:

```
CREATE TABLE DUMMY(A CHAR(1));
```

  Oracle hat schon eine DUMMY-Tabelle `DUAL`.

- Es ist wichtig, dass die Tabelle genau eine Zeile hat:

```
INSERT INTO DUMMY VALUES ('x');
```

- Dann können Sie die Funktion so testen:

```
SELECT UPPER('abc') FROM DUMMY;
```

  Bei manchen Datentypen zeigt das Standard-Format nicht alle Informationen über den Datenwert an. In Oracle müssen Sie z.B. `TO_CHAR(SQRT(2))` schreiben, wenn Sie alle Nachkommastellen sehen wollen.

Datentyp-Prädikate in SQL ausprobieren

- Prädikate werden in SQL klassisch in der `WHERE`-Bedingung verwendet.
- Wenn Sie eine `DUMMY`-Tabelle mit einer Zeile haben, können Sie den Wahrheitswert einer Bedingung z.B. so prüfen:

```
SELECT 'ja'
FROM   DUMMY
WHERE  'a' < 'A'
```

  Falls die Bedingung wahr ist, ist das Ergebnis `'ja'`, ansonsten ist es leer.

- Manche Systeme (u.a. PostgreSQL) erlauben Bedingungen auch direkt unter `SELECT` (als boolesche Ausdrücke).

  Dies ist inzwischen nach dem Standard erlaubt, aber nicht sehr portabel.
  Es geht z.B. nicht bei Oracle und MS SQL Server. Wenn man von der Logik her kommt, hält man die Trennung von Funktionen und Prädikaten für wichtig.

Inhalt

Zeichenketten (1)

`CHARACTER(n)`:

- Zeichenkette fester Länge mit n Zeichen.
- Daten, die in einer Spalte mit diesem Datentyp gespeichert werden, werden mit Leerzeichen bis zur Länge n aufgefüllt.

  Also wird immer Plattenspeicher für n Zeichen benötigt. Variiert die Länge der Daten stark, sollte man `VARCHAR` verwenden, siehe unten.
- `CHARACTER(n)` kann als `CHAR(n)` abgekürzt werden.
- Wird keine Länge angegeben, wird 1 angenommen.

  Somit erlaubt „`CHAR`“ (ohne Länge) das Speichern einzelner Zeichen.
  In Access scheint `CHAR` ohne Länge wie `CHAR(255)` behandelt zu werden.

Zeichenketten (2)

- Der Typ `CHAR(`*n*`)` war bereits im SQL-86-Standard enthalten.

  Natürlich wird er von allen normalen RDBMS unterstützt. Die Systeme unterscheiden sich darin, ob *n* in Bytes oder Zeichen gemessen wird.
  Nach dem Standard kann man `CHARACTERS` oder `OCTETS` hinter der Zahl angeben. Wenn nichts angegeben ist, sind es `CHARACTERS`. In realen DBMS sind es oft aber Bytes.

- Die Systeme unterscheiden sich im maximalen Wert für die Länge *n*.

DBMS	Maximales *n*	
Oracle 11	2000	Bytes
DB2 11.1	255	Bytes
SQL Server 2019	8000	Bytes
MySQL	255	Zeichen
PostgreSQL	10485760	Zeichen

Zeichenketten (3)

`VARCHAR(`*n*`)`:

- Zeichenkette variabler Länge mit bis zu *n* Zeichen.

  Es wird nur Speicherplatz für die tatsächliche Länge der Zeichenkette benötigt. Die maximale Länge *n* dient als Beschränkung, beeinflusst aber normalerweise das Dateiformat auf der Festplatte nicht. Wie bei `CHAR` ist systemabhängig, ob *n* in Bytes oder Zeichen gemessen wird. Es müssten eigentlich Zeichen sein, sind aber oft Bytes. Ggf. kann man die Einheit explizit wählen.

- Dieser Datentyp wurde im SQL-92-Standard hinzugefügt (im SQL-86-Standard nicht enthalten).

- Er wird jedoch wohl von allen modernen DBMS unterstützt.

  Er wird von allen in dieser Vorlesung besprochenen DBMS unterstützt (PostgreSQL, Oracle, DB2, SQL Server, Access, MySQL).

Zeichenketten (4)

- Offiziell heißt der Typ `CHARACTER VARYING(`*n*`)`, aber der Standard erlaubt die Abkürzung `VARCHAR`.
- Die Systeme unterscheiden sich im maximalen Wert für die maximale Länge *n*:

DBMS	Maximales *n*	
Oracle 11	4000	Bytes
DB2 11.1	32672	Bytes
SQL Server 2019	8000	Bytes
MySQL	65535	Bytes
PostgreSQL	10485760	Zeichen

Bei MySQL sind die Zeilen aber auf 65535 Bytes begrenzt. Vermutlich gibt es auch bei anderen Systemen Grenzen für die Zeilenlänge insgesamt. Außerdem sind bei MySQL Index-Einträge auf 767 Bytes begrenzt, lange Spalten können daher nicht als Schlüssel deklariert werden.

Zeichenkettenvergleich (1)

- Die Prädikate für Zeichenketten-Typen in SQL sind:
  - s_1 = s_2: s_1 ist gleich s_2.

    Man beachte, dass in vielen Systemen sich die Zeichenketten z.B. in Leerzeichen am Ende unterscheiden können, und trotzdem als „gleich“ gelten. Es ist also nicht die strenge mathematische Gleichheit.
  - s_1 <> s_2: s_1 ist verschieden von s_2.
  - s_1 < s_2: s_1 kommt in der Sortierreihenfolge vor s_2.
  - s_1 <= s_2: s_1 kommt vor s_2 oder ist gleich s_2.
  - s_1 > s_2: s_1 kommt in der Sortierreihenfolge nach s_2.
  - s_1 >= s_2: s_1 kommt nach s_2 oder ist gleich s_2.
  - s_1 LIKE s_2: s_1 passt auf das Muster s_2.
  - s_1 SIMILAR TO s_2: s_1 passt auf den regulären Ausdruck s_2.

Zeichenkettenvergleich (2)

- Das Ergebnis eines Vergleichs (=, <>, <, <=, >, >=) zweier Zeichenketten hängt von der explizit oder implizit gewählten „Collation“ (oder „Collation Sequence“) ab.

  Übersetzungen von „Collation“ sind u.a. „Vergleich“, „Zusammenstellung“.

- Nach dem SQL Standard kann man für jede Spalte einzeln Zeichensatz und Collation wählen, z.B.

```
NAME VARCHAR(20) CHARACTER SET UTF8
                 COLLATE UCS_BASIC
```

  Die verfügbaren Zeichensätze und Collations sind aber stark systemabhängig. Ein Zeichensatz kann verschiedene Collations erlauben. Im Normalfall sollte man für die ganze Datenbank den gleichen Zeichensatz wählen. Der Default wird beim Anlegen der Datenbank bzw. bei der Installation des DBMS festgelegt. Die genaue Vorgehensweise ist systemabhängig. Man teste mindestens deutsche Umlaute. Emojis, falls nötig, stellen besondere Anforderungen, weil sie im Unicode mehr als 16 Bit haben.

Zeichenkettenvergleich (3)

- `'a' < 'b'` usw. und `'A' < 'B'` usw. sollte immer gelten.
- Die voreingestellten Collations unterscheiden sich schon im Vergleich von Klein- und Großbuchstaben:
  - Z.B. gilt in Oracle und DB2: `'B' < 'a'` (binäre Sortierung).
  - In PostgreSQL liegen die Großbuchstaben zwischen den Kleinbuchstaben: `'a' < 'A' < 'b' < 'B'`.
  - SQL Server und MariaDB/MySQL sind case-insensitive, z.B. `'a' = 'A'`.
- Man kann aber natürlich jeweils explizit Collations wählen.

  Jeder Zeichensatz hat eine Default Collation, aber zu einem Zeichenvorrat kann es mehrere mögliche Collations geben. Bei manchen Systemen (z.B. PostgreSQL) kann man den Zeichensatz nur beim Anlegen der Datenbank wählen, aber die Collation für jede Spalte und jeden einzelnen Vergleich.

Zeichenkettenvergleich (4)

- Die simple Methode zum Vergleich zweier Zeichenketten s und t
  - basiert auf einer Ordnung der Zeichen und
  - vergleicht so lange jeweils das i-te Zeichen von s mit dem i-ten Zeichen von t, bis ein Vergleich nicht „=“ ergibt.
  - Dieser Vergleich gibt den Ausschlag.
- Wenn sich die Zeichenketten schon im ersten Zeichen unterscheiden, ist der Rest also egal.
- Mit dieses Verfahren kann aber z.B. folgende Ergebnisse von PostgreSQL nicht erklären:
  - `'b' < 'B'` : wahr
  - `'bz' < 'Ba'`: falsch

Zeichenkettenvergleich (5)

- Man kann beim Zeichenketten-Vergleich aber auch mehrere Durchläufe machen:
  - zuerst ein Vergleich ohne Berücksichtigung von Groß-/Kleinschreibung und Akzenten,
  - falls sich dabei kein Unterschied ergibt, ein Vergleich mit Akzenten, aber ohne Groß-/Kleinschreibung,
  - und ggf. schließlich noch ein Vergleich mit allen Details.

    Notfalls am Ende noch ein Vergleich mit den Binärcodes.

    [http://www.unicode.org/reports/tr10/tr10-41.html]
- PostgreSQL geht offensichtlich in mehreren Durchläufen vor.

  Die Details habe ich aber nicht in der Anleitung gefunden, Hinweise willkommen.
- Weitere Komplikation: „ß“ wie „ss“ behandeln.

Zeichenkettenvergleich (6)

- Für Zeichenketten veschiedener Länge gibt es
  - Non-Padded Vergleichs-Semantik: Z.B. `'a' < 'a '`.

    Strings werden Zeichen für Zeichen verglichen. Endet ein String und es wurde kein Unterschied gefunden, gilt der kürzere String als kleiner.
  - Blank-Padded Vergleichs-Semantik: Z.B. `'a' = 'a '`.

    Der kürzere String wird vor dem Vergleich mit `' '` aufgefüllt.
    Die Festlegung „`NO PAD`“ oder „`PAD SPACE`“ ist Teil der Collation.
- DB2, SQL Server, Access und MySQL verwenden blank-padded Semantik (zumindest als Default).
- PostgreSQL verwendet die blank-padded Semantik nur, wenn mindestens eine der Zeichenketten den Datentyp `CHARACTER(`n`)` hat (Konstanten haben nicht diesen Typ).

Zeichenkettenvergleich (7)

- Oracle hat non-padded Semantik, wenn mindestens ein Operand des Vergleichs den Typ `VARCHAR2` hat.

  Oracle hat einen Typ `VARCHAR2(`n`)` eingeführt. Er ist derzeit äquivalent zu `VARCHAR(`n`)`, aber Oracle beabsichtigt, die Vergleichs-Semantik für `VARCHAR` zu ändern, wobei die Semantik für `VARCHAR2` bleibt wie bisher. String-Konstanten in der Anfrage haben den Typ `CHAR(`n`)`. Z.B. kann ein Vergleich von `CHAR(10)`- und `CHAR(20)`-Spalten möglicherweise wahr sein, sowie ein Vergleich dieser Spalten mit z.B. `'abc'`. Aber `CHAR(10)` und `VARCHAR(20)` können nur gleich sein, wenn der `VARCHAR` zufällig 10 Zeichen hat. Leerzeichen am Stringende in `VARCHAR2`-Spalten sind ein Problem: unsichtbar in der Ausgabe, aber können „=“ verhindern.

LIKE

- Mit `LIKE` kann man auf einfache Muster vergleichen.
- In dem Muster (rechter Operand) steht
  - `%` für eine beliebige Folge beliebiger Zeichen.
  - `_` für ein einzelnes beliebiges Zeichen.
- Beispiel:

  ```
  EMAIL LIKE '%@acm.org'
  ```

  Die EMail-Adresse liegt in der Domain `acm.org`,
  d.h. `@acm.org` ist ein Suffix der Zeichenkette.
- Beispiel: Gesucht ist SQL als Teilstring im Aufgaben-Thema:

  ```
  THEMA LIKE '%SQL%'
  ```

- Für `LIKE` sind Leerzeichen am Ende signifikant.

Zeichenketten-Funktionen (1)

- s_1 || s_2: Konkatenation zweier Zeichenketten.

  Dies ist die Syntax im SQL-Standard seit SQL-92, und funktioniert in vielen Systemen, z.B. Oracle, PostgreSQL, DB2, SQLite. Aber es gibt leider wichtige Ausnahmen. Es funktioniert nicht in MS SQL Server, da muss man s_1 + s_2 schreiben. Es funktioniert auch nicht in MariaDB/MySQL, da muss man `CONCAT(`s_1`,`s_2`)` schreiben. Dies geht in MariaDB/MySQL auch mit mehr als zwei Strings, z.B. `CONCAT('a','b','c')`. PostgreSQL, Oracle und DB2 verstehen `CONCAT` auch, aber bei Oracle und DB2 nur mit zwei Argumenten.

- Beispiel: Tabellenzeile mit HTML-Tags erzeugen:

```
SELECT '<tr>' || '<td>' || ANR || '</td>' ||
       '<td>' || Punkte || '</td>' || '</tr>'
FROM   BEWERTUNGEN
WHERE  SID = 101 AND ATYP = 'H'
```

Zeichenketten-Funktionen (2)

- `CHARACTER_LENGTH(`*s*`)`, `CHAR_LENGTH(`*s*`)`:
  Länge einer Zeichenkette in Zeichen.

  Ist im SQL-Standard seit SQL-92. Wird verstanden in: PostgreSQL, MariaDB/MySQL, DB2 (ab Ver. 11, vorher zweiter Parameter für die Einheit nicht optional). Wird nicht verstanden in: Oracle, MS SQL Server. In Oracle schreibt man `LENGTH(`*s*`)` (wird auch in DB2, PostgreSQL, MariaDB/MySQL verstanden), in MS SQL Server `LEN(`*s*`)`.

- `OCTET_LENGTH(`*s*`)`: Länge in Bytes.

  Ist im SQL-Standard seit SQL-92. Wird verstanden in: PostgreSQL, MariaDB/MySQL, DB2. Wird nicht verstanden in: Oracle, MS SQL Server. In Oracle schreibt man `LENGTHB(`*s*`)`, in MS SQL Server `DATALENGTH(`*s*`)`.

- Beispiel: Länge des längsten Namens (für `MAX` siehe Kap. 13):

```
SELECT MAX(CHAR_LENGTH(NACHNAME))
FROM   STUDENTEN
```

Zeichenketten-Funktionen (3)

- `LOWER(s)`: Zeichenkette in Kleinbuchstaben umgewandelt.

  Ist im SQL-Standard seit SQL-92 (zusammen mit `UPPER` unter „fold“).
  Wird verstanden in: Oracle, PostgreSQL, MariaDB/MySQL, DB2,
  MS SQL Server. Viele Systeme verstehen alternativ auch `LCASE(s)`.

- `UPPER(s)`: Zeichenkette in Großbuchstaben umgewandelt.

  Ist im SQL-Standard seit SQL-92. Wird verstanden in: Oracle, PostgreSQL, MariaDB/MySQL, DB2, MS SQL Server. Viele Systeme verstehen alternativ auch `UCASE(s)`. Manche DBMS (u.a. Oracle) haben `INITCAP(s)`, was den ersten Buchstaben jedes Wortes in einen Großbuchstaben umwandelt.

- Beispiel: Case-Insensitiver Vergleich:

```
SELECT *
FROM   STUDENTEN
WHERE  UPPER(VORNAME) = UPPER('lisa')
```

  Natürlich könnte man rechts auch einfach `'LISA'` schreiben.

Zeichenketten-Funktionen (4)

- `POSITION(`s_1 `IN` s_2`)`:
  Position des ersten Vorkommens von s_1 in s_2.

  Ist im SQL-Standard seit SQL-92. Wird verstanden in: PostgreSQL, MariaDB/MySQL, DB2. Wird nicht verstanden in: Oracle, MS SQL Server. In Oracle schreibt man `INSTR(`s_2`,`s_1`)` (hier ist der Suchstring an zweiter Position). `INSTR(`s_2`,`s_1`,`p`,`n`)` beginnt die Suche an Position p und liefert das n-te Vorkommen. In MS SQL Server schreibt man für Suchstrings aus einem Zeichen `CHARINDEX(`s_1`,`s_2`)`, sonst `PATINDEX('%`s_1`%', `s_2`)`

- Die Positionen zählen von 1 ab.
- Wenn s_1 in s_2 nicht vorkommt, wird 0 geliefert.
- Beispiel: Position des @-Zeichens in der EMail-Adresse:

```
SELECT EMAIL, POSITION('@' IN EMAIL)
FROM   STUDENTEN
```

Zeichenketten-Funktionen (5)

- `SUBSTRING(`*s* `FROM` *p* `FOR` *n*`)`:
  Teilzeichenkette der Länge *n* von *s*, die an Position *p* beginnt.

  Z.B. liefert `substring('abcde',2,3)` das Ergebnis `'bcd'`.

  Ist im SQL-Standard seit SQL-92. Wird verstanden in: PostgreSQL, MariaDB/MySQL, DB2. Wird nicht verstanden in: Oracle, MS SQL Server.

  In Oracle schreibt man `SUBSTR(`*s*`,`*p*`,`*n*`)`. In MS SQL Server `SUBSTRING(`*s*`,`*p*`,`*n*`)`.

- `SUBSTRING(`*s* `FROM` *p*`)`: Ganzer Rest von *s* ab Position *p*.

  Wie oben. In Oracle schreibt man `SUBSTR(`*s*`,`*p*`)`. Beim MS SQL Server muss man ein hinreichend großes *n* wählen (ggf. mit `LEN` berechnen).

- Beispiel: Domain-Anteil der EMail-Adresse:

```
SELECT SUBSTRING(EMAIL FROM
                 POSITION('@' IN EMAIL)+1)
FROM   STUDENTEN
```

Zeichenketten-Funktionen (6)

- `TRIM(s)`: Teilzeichenkette von *s*, ohne Leerzeichen am Anfang oder Ende.

  Ist im SQL-Standard seit SQL-92. Wird verstanden in: PostgreSQL, Oracle, MariaDB/MySQL, DB2, MS SQL Server.

- `TRIM(LEADING c FROM s)`: Zeichen *c* am Anfang von *s* entfernen. Entsprechend `TRAILING` oder `BOTH`.

  Wenn man das Positions-Schlüsselwort weglässt, ist `BOTH` gemeint. Wenn man *c* weglässt, wird das Leerzeichen angenommen. Ist im SQL-Standard seit SQL-92. Wird verstanden in: PostgreSQL, MariaDB/MySQL, DB2. Wird nicht verstanden in: Oracle, MS SQL Server.
  In Oracle gibt es `LTRIM(s)`, `LTRIM(s,c)` `RTRIM(s)`, `RTRIM(s,c)`.
  In MS SQL Server `LTRIM(s)`, `RTRIM(s)` und `TRIM(c FROM s)`.
  Nach dem SQL Standard ist es ein Fehler, wenn *c* nicht genau ein Zeichen enthält. Viele Systeme erlauben es und verhalten sich aber unterschiedlich (beliebige Zeichen aus *c* oder nur vollständiger String wird entfernt).

Zeichenketten-Funktionen (7)

- Leerzeichen am Ende sind bei der Ausgabe „unsichtbar“.

  In einigen Systemen (z.B. Oracle) sind sie aber für den Vergleich wichtig.
  Andere Systeme ignorieren beim Vergleich Leerzeichen am Ende.

- Man kann Leerzeichen z.B. auf folgende Art sichtbar machen:

```
SELECT '"' || NACHNAME || '"'
FROM   STUDENTEN
```

- Man kann sie auch für den Vergleich entfernen:

```
SELECT *
FROM   STUDENTEN
WHERE  TRIM(NACHNAME) = 'Weiss'
```

- Am besten entfernt man sie vor der Einfügung in die DB.

Zeichenketten-Funktionen (8)

Einige weitere Funktionen (nicht im Standard):

- RPAD(*s*,*n*): Auffüllen mit Leerzeichen bis zur Länge *n*.

  Wenn *s* länger als *n* ist, wird der Rest abgeschnitten!

  LPAD(*s*,*n*): Das gleiche auf der linken Seite (vorne).

- SOUNDEX(*s*): String, der den Klang von *s* codiert.

  Man könnte mit `SOUNDEX(NACHNAME) = SOUNDEX('Meier')` verschiedene Schreibvarianten finden, wenn es für deutsche Aussprache gemacht wäre.

- ASCII(*c*)/UNICODE(*c*)/ORD(*c*): Code des Zeichens *c*.
- CHR(*n*)/CHAR(*n*): Zeichen mit Code *n*.
- TRANSLATE(*s*,*x*,*y*): Ersetze in *s* das *i*-te Zeichen von *x* durch das *i*-te Zeichen von *y* (bzw. lösche es, falls *y* kürzer).
- REPLACE(*s*,*x*,*y*): Ersetze Zeichenkette *x* durch *y* in *s*.

Weitere Informationen

- PostgreSQL
  [https://www.postgresql.org/docs/9.1/functions-string.html]
- Oracle
  [https://docs.oracle.com/database/121/SQLRF/functions002.htm]
- MySQL
  [https://dev.mysql.com/doc/refman/8.0/en/string-functions.html]
- Microsoft SQL Server
  [https://docs.microsoft.com/en-us/sql/t-sql/functions/string-functions-transact-sql]
- IBM DB2
  [https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/com.ibm.db2.luw.sql.ref.doc/doc/c0000767.html]
- Vergleich
  [https://en.wikibooks.org/wiki/SQL_Dialects_Reference/Functions_and_expressions/String_functions]

Inhalt

Festkomma-Zahlen (1)

- `NUMERIC(p,s)`: Vorzeichenbehaftete Zahl mit insgesamt p Ziffern (s Ziffern hinter dem Komma).

  Wird auch Festkommazahl/Fixpunktzahl genannt, da das Komma immer an der gleichen Stelle steht (im Gegensatz zu Gleitkommazahlen).

- Z.B. erlaubt `NUMERIC(3,1)` die Werte `-99.9` bis `99.9`.

  MySQL erlaubt Werte von `-99.9` bis `999.9` (falsch).

- `NUMERIC(p)`: Ganze Zahl mit p Ziffern.

  `NUMERIC(p)` ist das gleiche wie `NUMERIC(p,0)`. „`NUMERIC`“ ohne p verwendet ein implementierungsabhängiges p.

- „`NUMERIC(p[,s])`“ war bereits in SQL-86 enthalten.

  Es wird nicht in Access unterstützt, aber in den anderen vier DBMS.

Festkomma-Zahlen (2)

- `DECIMAL(p,s)`: fast das Gleiche wie `NUMERIC(p,s)`.

  Hier sind größere Wertemengen möglich. Z.B. muss das DBMS bei `NUMERIC(1)` einen Fehler ausgeben, wenn man versucht, 10 einzufügen. Bei `DECIMAL(1)` kann das DBMS evtl. den Wert speichern (wenn sowieso ein ganzes Byte für die Spalte verwendet wird). Übrigens gibt MySQL nie einen Fehler aus, es nimmt einfach den größtmöglichen Wert.

- `DECIMAL` kann mit „`DEC`“ abgekürzt werden.
- Wie `NUMERIC` gab es auch `DECIMAL` schon in SQL-86.
- Oracle verwendet `NUMBER(p,s)` und `NUMBER(p)`, versteht aber auch `NUMERIC`/`DECIMAL` als Synonyme.

  Keines der anderen vier Systeme versteht `NUMBER`.

Festkomma-Zahlen (3)

- Die Präzision p (Gesamtanzahl der Ziffern) kann zwischen 1 und einem gewissen Maximum liegen.

DBMS	Maximales p	
Oracle 8.0	38	
DB2 UDB 5	31	
SQL Server 7	28/38	
MySQL	253/254	(arith. ca. 15)
PostgreSQL	1000	

In SQL Server muss der Server mit der Option `/p` gestartet werden, um bis zu 38 Ziffern zu unterstützen (sonst 28). MySQL speichert `NUMERIC(`p`,`s`)` als String von p Ziffern und Zeichen für „-“, „.“. Aber MySQL macht Berechnungen mit `DOUBLE` (ca. 15 Ziffern Genauigkeit).

- Der Parameter s muss $s \geq 0$ und $s \leq p$ erfüllen.

In Oracle muss $-84 \leq s \leq 127$ gelten (egal, wie groß p ist).

Festkomma-Zahlen (4)

- `INTEGER`: Vorzeichenbehaftete ganze Zahl, dezimal oder binär gespeichert, Wertebereich ist implementierungsabhängig.

  PostgreSQL, DB2, SQL Server, MySQL und Access verwenden 32 Bit-Binärzahlen: -2 147 483 648 (-2^{31}) .. +2 147 483 647 ($2^{31}-1$). D.h. der Wertebereich in diesen DBMS ist etwas größer ist als `NUMERIC(9)`, aber der SQL-Standard garantiert dies nicht.
  In Oracle: Synonym für `NUMBER(38)`.

- `INT`: Abkürzung für `INTEGER`.

- `SMALLINT`: Wie oben, Wertebereich evtl. kleiner.

  PostgreSQL, DB2, SQL Server, MySQL und Access verwenden 16 Bit-Binärzahlen: -32 768 (-2^{15}) .. +32 767 ($2^{15}-1$). Somit ist der Bereich in diesen Systemen größer als `NUMERIC(4)`, aber kleiner als `NUMERIC(5)`.
  In Oracle wieder ein Synonym für `NUMBER(38)`.

Festkomma-Zahlen (5)

- Zusätzliche, nicht standardisierte Integertypen:
  - `BIT`: In SQL Server (0,1), Access (-1, 0).

    In MySQL gelten `BIT` und `BOOL` als Synonyme für `CHAR(1)`.
  - `TINYINT`: In MySQL (-128 .. 127), in SQL Server (0 .. 255).

    In Access kann der Typ `BYTE` die Werte 0 .. 255 speichern.
  - `BIGINT`: In DB2 und MySQL (-2^{63} .. $2^{63}-1$).

    Der Wertebereich ist größer als `NUMERIC(18)`.
  - MySQL unterstützt z.B. auch `INTEGER UNSIGNED`.

    In MySQL kann man auch eine Ausgabe-Weite definieren (z.B. `INTEGER(5)`) und `ZEROFILL` hinzufügen, um festzulegen, dass z.B. 3 als 0003 dargestellt wird, wenn die Ausgabe-Weite 4 ist.

Festkomma-Zahlen (6)

`INT` vs. `NUMERIC`:

- Wenn die Spezifikation eine bestimmte Anzahl Dezimalziffern verlangt, geht eigentlich nur `NUMERIC`:
  - Der Wertebereich von `INT` ist implementierungsabhängig, obwohl 32 Bit sehr typisch ist.

    Das maximale p in `NUMERIC(`p`)` ist auch implementierungsabhängig, aber man bekommt wenigstens eine Fehlermeldung, wenn es nicht reicht.
- In vielen Systemen verbraucht `INT` weniger Speicherplatz als `NUMERIC`, und Rechnungen mit `INT` gehen schneller.
- Eventuell wäre der Typ `INT` zusammen mit einem `CHECK`-Constraint ($\rightarrow$ Sommer) für das Intervall eine Option.

  Die Größe des Wertebereichs (nützliche Information für Optimierer) und die benötigte Ausgabebreite wären bei `NUMERIC(`p`)` klarer.

Hinweis zur Division

- Einige Systeme machen eine „Integer Division“, wie etwa aus Java bekannt, wenn man ganze Zahlen teilt.

  Das Ergebnis der Division ist dann wieder eine ganze Zahl.

- Z.B. ist dann $3/2 = 1$.

  Diese `WHERE`-Bedingung ist wahr in: PostgreSQL, DB2, MS SQL Server.
  Sie ist falsch in: Oracle, MariaDB/MySQL.

- Oft macht es einen Unterschied, ob eine Spalte vom Typ `NUMERIC(`n`)` oder vom Typ `INT` am Vergleich beteiligt ist.

  Z.B. Spalte `A` vom Typ `NUMERIC(1)` und `B` vom Typ `INT`. Beide enthalten den Wert 3. Bei PostgreSQL, DB2 und MS SQL Server ist `A/2=1.5` und `B/2=1`. Bei Oracle und MariaDB/MySQL sind beide Ergebnisse `1.5`.

- By „Integer Division“ gibt es eine Exception für Division durch 0! (Bei `WHERE` keine Auswertungsreihenfolge garantiert.)

Gleitkomma-Zahlen (1)

- `FLOAT(`p`)`: Gleitkommazahl $M * 10^E$ mit mindestens p Bits Präzision für M ($-1 < M < +1$).
- `REAL`, `DOUBLE PRECISION`: Abkürzungen für `FLOAT(`p`)` mit implementierungsabhängigen Werten für p.
- Z.B. SQL Server (DB2 und MySQL ähnlich):
  - `FLOAT(`p`)`, $1 \leq p \leq 24$, verwendet 4 Bytes.

    7 Ziffern Präzision (Wertebereich `-3.40E+38` bis `3.40E+38`).

    `REAL` bedeutet `FLOAT(24)`.
  - `FLOAT(`p`)`, $25 \leq p \leq 53$, verwendet 8 Bytes.

    15 Ziffern Präzision (Wertebereich `-1.79E+308` bis `+1.79E+308`).

    `DOUBLE PRECISION` bedeutet `FLOAT(53)`.

Gleitkomma-Zahlen (2)

- Oracle verwendet `NUMBER` (ohne Parameter) als Datentyp für Gleitkommazahlen.

  Oracle versteht auch `FLOAT(`p`)`. `NUMBER` erlaubt das Speichern von Werten zwischen $1.0 * 10^{-130}$ und $9.9\ldots * 10^{125}$ mit 38 Ziffern Präzision.

- Access versteht `REAL`, `FLOAT` (ohne Parameter) und `DOUBLE` (ohne Schlüsselwort `PRECISION`).

- `NUMERIC`, `DECIMAL` etc. sind exakte numerische Datentypen. `FLOAT` ist ein gerundeter numerischer Typ: Rundungsfehler sind nicht wirklich kontrollierbar.

  Z.B. sollte man für Geld nie `FLOAT` verwenden.

Prädikate für Zahlen

- Die Prädikate für Zahl-Datentypen in SQL sind:
  - x = y: x ist gleich y ($x = y$).

    Man beachte, dass Gleitkomma-Zahlen immer mit einer gewissen Ungenauigkeit behaftet sind (sie heißen ja auch „approximate numeric types“). Ein Vergleich auf Gleichheit oder Ungleichheit ist daher problematisch. Man testet bei diesen Typen besser auf ein kleines Intervall um den wahren Wert.
  - x <> y: x ist verschieden von y ($x \neq y$).
  - x < y: x ist kleiner als y ($x < y$).
  - x <= y: x is kleiner oder gleich y ($x \leq y$).
  - x > y: x ist größer als y ($x > y$).
  - x >= y: x is größer oder gleich y ($x \geq y$).

Funktionen für Zahlen (1)

- Im SQL-86/89 Standard gab es nur die vier Grundrechenarten: $x + y$, $x - y$, $x * y$, x / y.

  Plus und Minus können auch monadisch verwendet werden: $+x$. $-x$.

- In SQL-92 änderte sich daran nichts.

  „Numeric value functions“: `CHAR_LENGTH`, `OCTET_LENGTH`, `POSITION` (alle für Zeichenketten), `EXTRACT` (Zugriff auf Komponenten von Datums/Zeitwerten).

- SQL-99 brachte zwei neue Funktionen:

  - `ABS(`x`)`: Absolutwert von x.

    Funktioniert in: PostgreSQL, Oracle, MariaDB/MySQL, DB2, MS SQL.

  - `MOD(`n`,`m`)`: Rest bei Division von n durch m.

    Funktioniert in: PostgreSQL, Oracle, MariaDB/MySQL, DB2.

    Funktioniert nicht in: MS SQL Server. Dort schreibt man n % m.

    Das verstehen auch PostgreSQL, MySQL (nicht Oracle, DB2).

Funktionen für Zahlen (2)

- SQL 2003 hatte folgende neuen Funktionen:
  - `LN(x)`: Natürlicher Logarithmus: $\ln(x)$.

    In MS SQL Server: `LOG`.
  - `EXP(x)`: Exponentialfunktion: e^x.
  - `POWER(x,y)`: Potenz: x^y.
  - `SQRT(x)`: Quadratwurzel: $\sqrt{x}$.
  - `FLOOR(x)`: Abrundung: $\lfloor x \rfloor$.
  - `CEILING(x)`, `CEIL(x)`: Aufrundung: $\lceil x \rceil$.

    In Oracle nur `CEIL`, in MS SQL Server nur `CEILING`.
  - `WIDTH_BUCKET(x,y,z,n)`: In welchem von n Teilintervallen des Intervalls $[y, z]$ liegt x?

    Falls $x < y$, ist das Ergebnis 0. Falls $x \geq z$, ist das Ergebnis $n + 1$.

    Das Ergebnis 1 bedeutet z.B., dass $y \leq x < y + (z - y)/n$.

    Funktioniert in PostgreSQL, Oracle, DB2. Nicht MySQL, MS SQL.

Funktionen für Zahlen (3)

- Erst SQL-2016 brachte:
  - `SIN(`x`)`, `COS(`x`)`, `TAN(`x`)`, `SINH(`x`)`, `COSH(`x`)`, `TANH(`x`)`, `ASIN(`x`)`, `ACOS(`x`)`, `ATAN(`x`)`: Trigonometrische Funktionen.

    Winkel werden in Bogenmaß (`rad`) gemessen: $180° = \pi = 3.14159$.
    PostgreSQL, MariaDB/MySQL und MS SQL Server kennen nicht `SINH`, `COSH`, `TANH`. Die anderen Funktionen sind problemlos.
  - `LOG(`b`,`x`)`: Logarithmus zur Basis b: $\log_b(x)$.

    Bei MS SQL Server sind die Argumente vertauscht: `LOG(`x`,`b`)`.
    DB2 kennt `LOG` nur mit einem Argument, und dann ist es der Logarithmus zur Basis $e = 2.71828$.
  - `LOG10(`x`)`: Logarithmus zur Basis 10: $\log_{10}(x)$.

    PostgreSQL und Oracle kennen nicht `LOG10`. Bei PostgreSQL liefert `LOG(`x`)` den Logarithmus zur Basis 10.

Funktionen für Zahlen (4)

- Viele der mathematischen Funktionen gab es in realen DBMS schon weit vor der Aufnahme in den SQL Standard.
- Viele Systeme haben auch:
  - `ROUND(`x`)`: x auf die nächste ganze Zahl gerundet.
  - `ROUND(`x`,`n`)`: x auf n Stellen nach dem Komma gerundet.
  - `DEGREES(`x`)`, `RADIANS(`x`)`: Umrechnung zwischen Grad und Bogenmaß.
  - `ATAN2(`x`,`y`)`: Polarwinkel des Punktes (x, y).
  - `PI()`: π (Kreiszahl).
  - `RAND()`: Zufallszahl.

Weitere Informationen

- PostgreSQL

  [https://www.postgresql.org/docs/10/functions-math.html]
- Oracle (wie oben, nur eine Seite für alle Funktionen)

  [https://docs.oracle.com/database/121/SQLRF/functions002.htm]
- MySQL

  [https://dev.mysql.com/doc/refman/8.0/en/numeric-functions.html]
- Microsoft SQL Server

  [https://docs.microsoft.com/en-us/sql/t-sql/functions/mathematical-functions-transact-sql]
- IBM DB2 (wie oben)

  [https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/...]
- Vergleich

  [https://en.wikibooks.org/wiki/SQL_Dialects_Reference/Math_functions/Numeric_functions]

  [http://en.wikibooks.org/.../Math_functions/Trigonometric_functions]

Datentypen in SQL-86

- CHAR[ACTER][(*n*)] [. . .] markiert optionale Teile.
- NUMERIC[(*p*[,*s*])]
- DEC[IMAL][(*p*[,*s*])]
- INT[EGER], SMALLINT
- FLOAT[(*p*)], REAL, DOUBLE PRECISION
- Diese Typen sollten sehr portabel sein.

  Access unterstützt NUMERIC, DECIMAL, FLOAT(*p*), DOUBLE PRECISION nicht.
- Alle untersuchten Systeme unterstützen zusätzlich VARCHAR, was nicht im SQL-86-Standard enthalten war.

Inhalt

Lange Zeichenketten (1)

- Die maximale Länge von `VARCHAR` is üblicherweise stark beschränkt (systemabhängig, z.B. 4000 Zeichen).
- Will man ganze Textdateien als Tabelleneinträge erlauben, so verwendet man den Typ `CLOB` („Character Large Object“).

  Man kann optional eine Maximallänge angeben, z.B. `CLOB(16M)`.
- Die Verwendung dieser Datentypen in Anfragen ist oft stark eingeschränkt.

  Z.B. kann man oft nicht in Bedingungen verwenden, also z.B. nicht mit `LIKE` nach Teilzeichenketten suchen. Auch Duplikateliminierung oder Sortierung wird nicht unterstützt. Ebenso String-Konkatenation. Man kann nur die Datei in der Datenbank abspeichern, und sie wieder herausholen, oft nur über eine Programmierschnittstelle. Die genauen Einschränkungen sind systemabhängig.

Lange Zeichenketten (2)

- Wenn man eine Suche mit `LIKE` ermöglichen will, wäre eine Lösung, Zeilen einzeln abzuspeichern.
- In einigen DBMS heißt der Datentyp für lange Strings „`TEXT`“.

  In MySQL erlaubt `TEXT` nur 64 KByte, es gibt aber `MEDIUMTEXT` bis 16 MByte und `LONGTEXT` bis 4 GByte.

  In MS SQL Server erlaubt `TEXT` bis 2 GByte. Man kann auch `VARCHAR(MAX)` schreiben für Zeichenketten bis 2 GByte Länge.
- Natürlich könnte man in der Datenbank auch nur den Dateinamen speichern, und die eigentlichen Daten außerhalb der Datenbank in normalen Betriebssystem-Dateien.
- Dann verliert man aber die Transaktionsverwaltung der Datenbank (z.B. Backup/Recovery) und es besteht die Gefahr von „Broken Links“.

Binäre Daten

- Bei Oracle findet eine automatische Zeichensatz-Konvertierung zwischen Client und Server statt. Für Textdaten ist das recht nützlich, aber binäre Daten werden dadurch zerstört.
- Man muss deswegen einen Datentyp zu wählen, bei dem das DBMS versteht, dass es sich um binäre Daten handelt.
- Der SQL Standard hat die Typen `BINARY`, `BINARY(`*n*`)`, `VARBINARY(`*n*`)`, `BLOB`, `BLOB(`*n*`)`.

  `BLOB` steht für „Binary Large Object“. Oracle hat auch den Typ `RAW(n)` für kurze Bytefolgen fester Länge. Einige Systeme verwenden einen Zusatz zu einem Zeichen-Datentyp, z.B. DB2: „`VARCHAR(100) FOR BIT DATA`“, MySQL: „`CHAR(`*n*`) BINARY`“, „`VARCHAR(`*n*`) BINARY`“. MS SQL Server kennt `BINARY(`*n*`)`, `VARBINARY(`*n*`)`, sowie `VARBINARY(MAX)` als BLOB-Typ (2 GByte). Konstanten können in vielen System hexadeximal geschrieben werden, z.B. `X'FFFF'` in DB2 (so auch im Standard) oder `0xFFFF` in MySQL.

Boolesche Werte

- Der SQL-92-Standard hatte keinen booleschen Datentyp.

  Z.B. Oracle 11g kennt auch kein `BOOLEAN`.

- Der Typ `BOOLEAN` wurde erst in SQL-99 eingeführt.

  Einige moderne Systeme, z.B. PostgreSQL, verstehen `BOOLEAN`.
  MS SQL Server hat den Typ `BIT`. MySQL versteht `BOOLEAN` als Abkürzung für `TINYINT`, und erlaubt daher die Werte -128 bis $+127$.

- Die möglichen Werte sind `TRUE` und `FALSE`,
  plus (falls nicht ausgeschlossen) der Nullwert, für den es auch die Konstante `UNKNOWN` (vom Typ `BOOLEAN`) gibt.

- Klassisch wird meist der Typ `CHAR(1)` verwendet, zusammen mit der Integritätsbedingung, dass in dieser Spalte nur `'J'` und `'N'` erlaubt sind.

Datumsangaben

- Datums-, Zeit- und Intervall-Datentypen sind systemabhängig.
- Viele Systeme haben einen Typ `DATE` für Datumsangaben.
  - Oft können die Werte als Strings im Format `'YYYY-MM-DD'` eingegeben werden.
  - SQL-92 verlangt ein Typ-Schlüsselwort: `DATE '2019-03-30'`

    In Oracle enthält `DATE` auch eine Uhrzeit, und das String-Format ist abhängig von den landesspezifischen Einstellungen.
- Es gibt auch die Typen `TIME` für Uhrzeiten und `TIMESTAMP` für eine Kombination aus Datum und Uhrzeit.

  Im SQL Standard kann man Nachkommastellen für die Sekunden festlegen, sowie die Speicherung einer Zeitzone verlangen, z.B. `TIME(3) WITH TIME ZONE`. Bei SQL Server ist `TIMESTAMP` eine eindeutige Nummer, und man muss `DATETIME` verwenden. `TIME` hat dort immer Nachkommastellen.

Zeichenketten in nationalem Zeichensatz

- Bei manchen Systemen wird der Datenbank-Zeichensatz bei der Installation des DBMS festgelegt, und kann anschließend nicht geändert werden.

  Man sollte also rechtzeitig prüfen, dass man alle benötigten Zeichen speichern kann. Allerdings darf man bei modernen Systemen hoffen, dass der Default Unicode z.B. in der UTF-8 Codierung ist.

- Nach dem SQL-Standard (und z.B. in MySQL) kann man Zeichensatz und Sortierreihenfolge für jede Spalte einzeln festlegen.

- Nach dem SQL-Standard gibt es String-Typen mit einem implementierungsabhängigen „nationalen“ Zeichensatz.

  `NATIONAL CHARACTER(`*n*`)` oder kurz `NCHAR(`*n*`)` und `NCHAR VARYING(`*n*`)`.
  Konstanten werden z.B. `N'abc'` geschrieben. Teils wird das dann für Unicode verwendet, falls der normale Zeichensatz nicht Unicode ist.

Weitere Datentypen

- Referenzen zu externen Dateien (URIs).

  Oracle: `BFILE`, DB2: `DATALINK`.

- Physischer Zeiger auf eine bestimmte Zeile.

  Oracle: `ROWID`.

- Geldbeträge

  SQL Server: `MONEY`, `SMALLMONEY`. Access: `CURRENCY`.

- Eindeutige Zahlen.

  SQL Server: `TIMESTAMP`, `UNIQUEIDENTIFIER`. Access: `GUID`.

- Aufzählungstypen und Mengen.

  MySQL: `ENUM`$(v_1, v_2, \ldots)$, `SET`$(v_1, v_2, \ldots)$.