Einführung in Datenbanken

Kapitel 3: SQL: Lexikalische Syntax

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2024/25

http://www.informatik.uni-halle.de/~brass/db24/

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Syntax-Graphen lesen können.

  Also die beschriebene Syntax mit eigenen Worten erklären, sowie die syntaktische Korrektheit von Zeichenfolgen bezüglich eines Syntaxgraphen beurteilen.

- Wissen, was es bedeutet, dass SQL eine formatfreie Sprache ist (und davon bei eigenen Anfragen Gebrauch machen).
- Kommentare in SQL schreiben.
- Zahl- und Zeichenkettenkonstanten in SQL schreiben.
- Bezeichner (Namen) in SQL schreiben.
- „Delimited Identifier“ („begrenzte Bezeichner“) verwenden und von String-Konstanten unterscheiden.

Inhalt

Syntax-Formalismus

- In dieser Vorlesung wird die Syntax von SQL-Anfragen mit „Syntax-Graphen“ definiert.

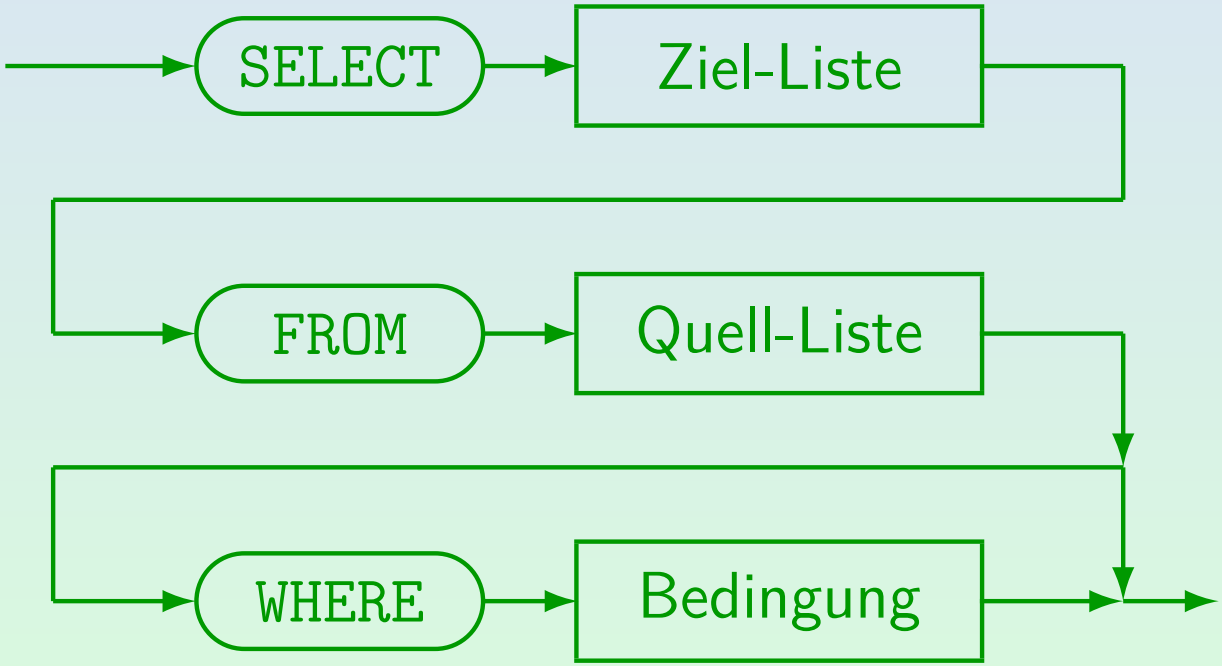
  Beispiel auf nächster Folie. Alternative zu kontextfreien Grammatiken (definiert dieselbe Klasse von Sprachen). In Anhang D genauer.

- Um eine Zeichenfolge syntaktisch richtig zu erstellen, muss man einen Pfad vom Start bis zum Ziel durch den Graphen verfolgen.

  Wörter in Ovalen werden direkt in die Ausgabe geschrieben, Kästen sind „Aufrufe“ anderer Graphen: An diesem Punkt muss man einen Pfad durch den Graphen finden, dessen Name in dem Kasten steht. Anschließend kehrt man zu der Kante zurück, die den Kasten verlässt.
  Das Oracle-SQL-Referenz-Handbuch enthält auch Syntax-Graphen, aber dort ist die Bedeutung von Ovalen und Kästen umgekehrt.

Basis-Anfrage-Syntax (1)

SELECT-Ausdruck (vereinfacht):



Basis-Anfrage-Syntax (2)

- Nach dem Syntaxdiagramm (und dem SQL Standard) muss jede SQL-Anfrage die Schlüsselwörter `SELECT` und `FROM` enthalten. Dagegen ist der `WHERE`-Teil optional.

  Oracle stellt eine Dummy-Relation „`DUAL`“ zur Verfügung, die nur eine Zeile hat. Sie kann benutzt werden, wenn nur eine Berechnung ohne Zugriff auf die DB durchgeführt wird: „`SELECT TO_CHAR(SQRT(2)) FROM DUAL`“ berechnet $\sqrt{2}$.

- Es gibt die SQL-Grammatik aus dem Standard online:

  [https://jakewheat.github.io/sql-overview/sql-2016-foundation-grammar.html#query-specification]

  Man kann auch bei der größeren Struktur query-expression einsteigen,

- In PostgreSQL, SQL Server, Access und MySQL kann man die `FROM`-Klausel weglassen, und z.B. `SELECT 1+1` schreiben.

  In Oracle, DB2 und dem SQL-92-Standard ist dies ein Syntax-Fehler.

Basis-Anfrage-Syntax (3)

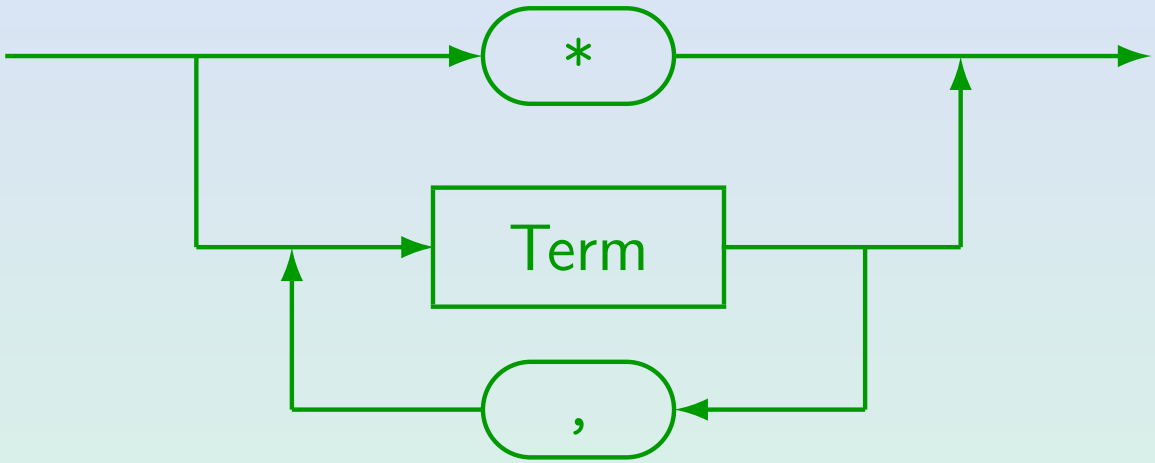
Anmerkung zur Portabilität:

- Sie werden in der Klausur nicht gefragt, bei welchen Systemen darf man die `FROM`-Klausel weglassen?
- Wenn Sie ad-hoc Anfragen schreiben (Sie interessieren sich nur für das Ergebnis und bewahren die Anfrage nicht auf), spielt Portabilität keine Rolle.
- Wenn Sie Anfragen in Programmen schreiben, schon.

  Es gibt immer die Möglichkeit, dass man später auf ein anderes DBMS wechseln muss. Für Portabilitätsprobleme sollte man zumindest einen Vorteil bekommen (z.B. kürzere Anfrage, besonders effizient implementiertes Spezialkonstrukt). Im konkreten Beispiel kann man auf eine eigene Dummy-Tabelle verzichten.
- Unnötige Verletzungen der Portabilität können in der Klausur zu (kleinen) Punktabzügen führen.

Basis-Anfrage-Syntax (4)

Ziel-Liste (vereinfacht):



- Man kann also
  - entweder ein "*" schreiben (zur Ausgabe aller Spalten),
  - oder eine Liste von auszugebenden Termen (s.u., das sind insbesondere Spalten) angeben (durch Kommata getrennt).
- Beispiel: `SELECT VORNAME, NACHNAME FROM STUDENTEN`

Basis-Anfrage-Syntax (5)

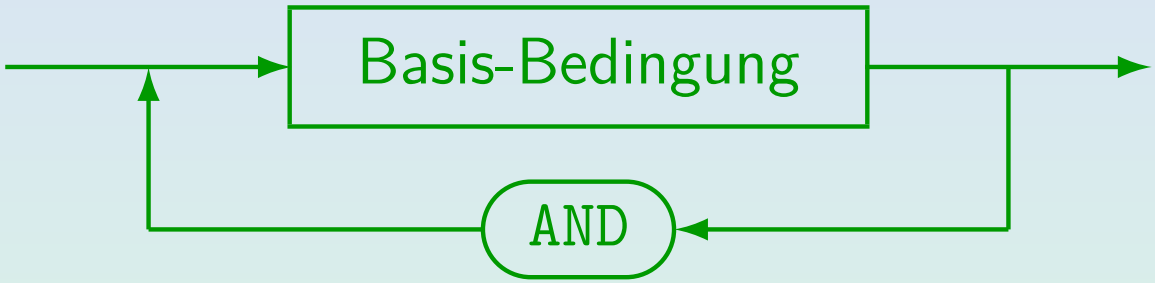
Quell-Liste (vereinfacht):

- In SQL kann man unter `FROM` eigentlich auf mehrere Tabellen zugreifen.
- Das wird später ausführlich behandelt.

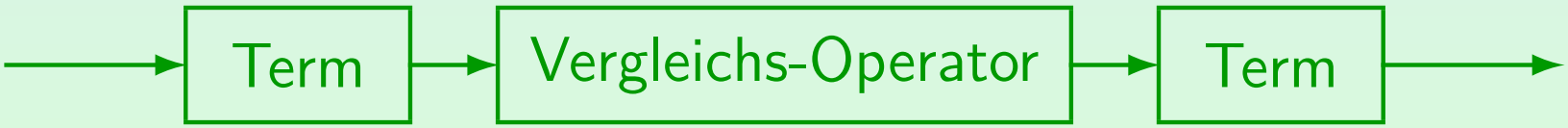
  Hier ist nur eine Grammatik für eine ganz einfache SQL-Teilmenge gezeigt. Eigentlich ist das Ziel nur, den Syntax-Formalismus zu erklären. Es ist aber auch für die Übungen praktisch, schon einfache Anfragen stellen zu können.

Basis-Anfrage-Syntax (6)

Bedingung (vereinfacht):

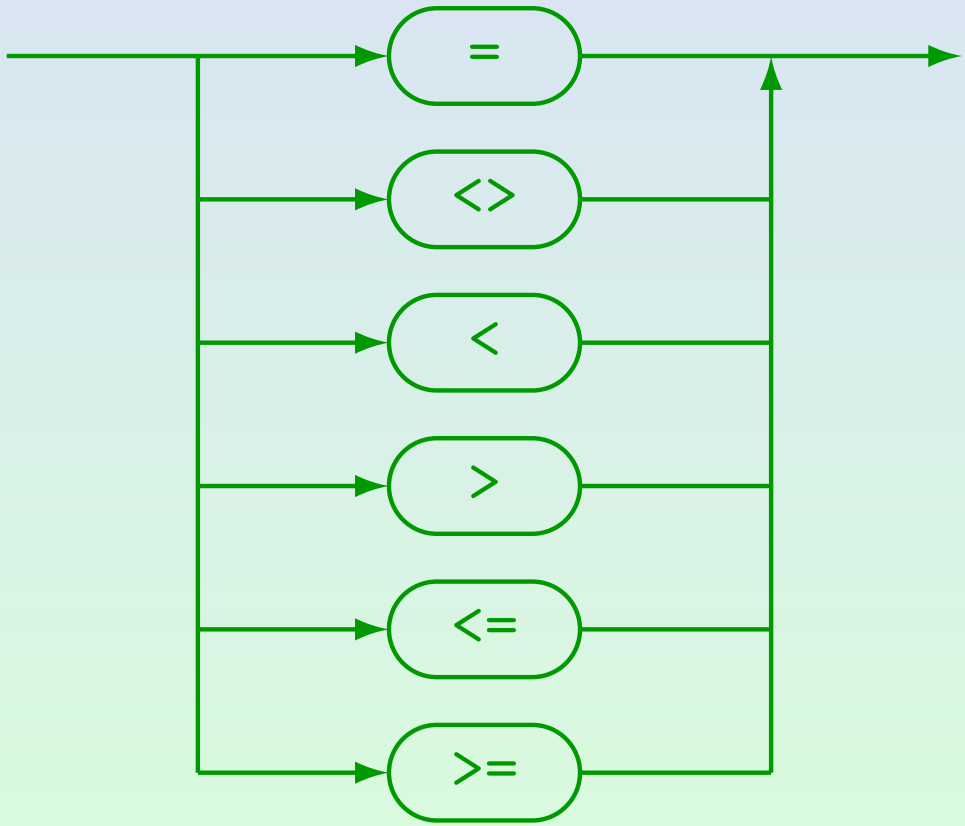


Basis-Bedingung (vereinfacht):



Basis-Anfrage-Syntax (7)

Vergleichs-Operator:



Basis-Anfrage-Syntax (8)

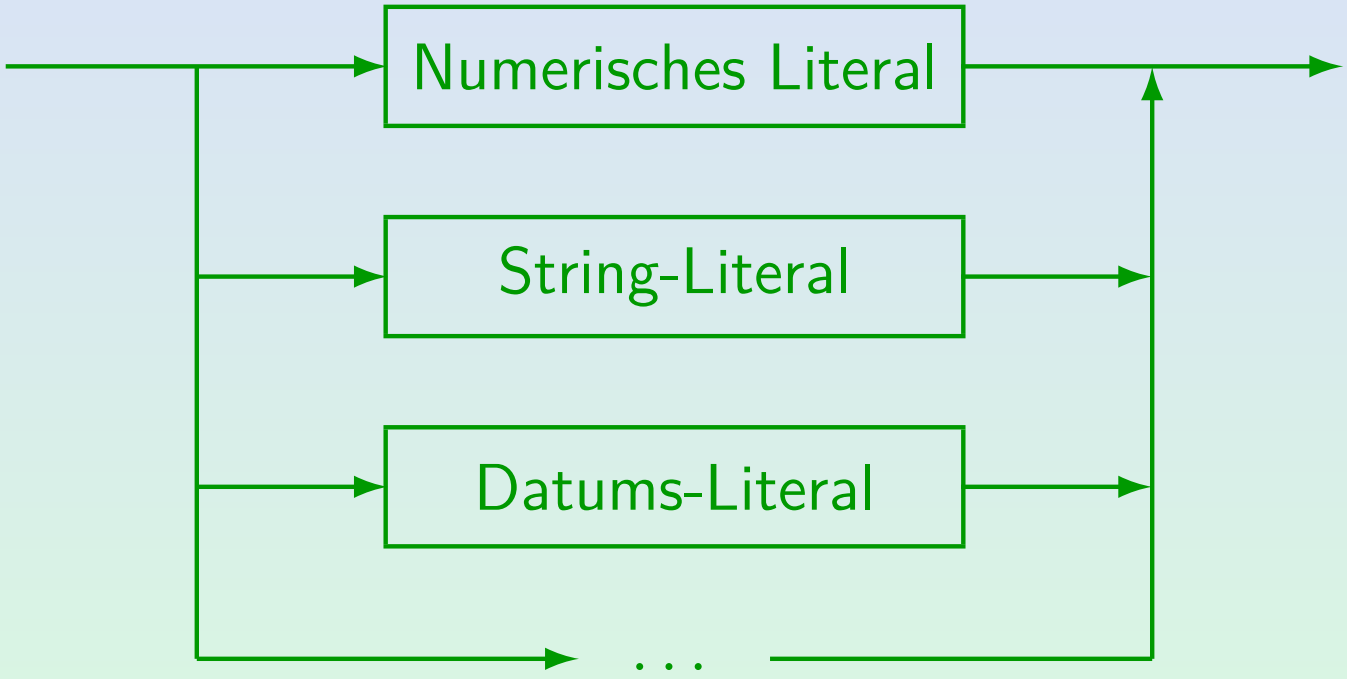
Term (vereinfacht):

Tabelle:

Spalte:

Basis-Anfrage-Syntax (9)

Konstante (Literal):



Man spricht bei Programmiersprachen üblicherweise von „Literalen“ statt Konstanten, um direkt aufgeschriebene Datenwerte von symbolischen Konstanten zu unterscheiden. Es gibt in SQL aber (fast) keine symbolischen Konstanten (nur z.B. `current_date`, `current_user` — dies sind natürlich auch Terme).

Inhalt

Lexikalische Syntax

- Die lexikalische Syntax einer Sprache definiert, wie Wortsymbole („Token“) aus einzelnen Zeichen zusammengesetzt werden.
- Z.B. definiert sie die genaue Syntax von
  - Bezeichnern (Namen für z.B. Tabellen, Spalten),
  - Literalen (Datentyp-Konstanten, z.B. Zahlen),
  - Schlüsselwörtern, Operatoren, Trennzeichen.
- Anschließend wird die Syntax von Anfragen u.s.w. basierend auf diesen Wortsymbolen definiert.

  D.h. eine Anfrage wird dann als Folge von Token definiert, nicht als Folge von Zeichen. Diese Zweiteilung bewirkt z.B., dass man auf der höheren Syntaxebene nicht mehr über eingestreuten Leerplatz nachdenken muss.

Leerzeichen und Kommentare

Leerplatz ist zwischen Wortsymbolen (Token) erlaubt:

- Leerzeichen (meist auch Tabulator-Zeichen)
- Zeilenumbrüche
- Kommentare:
  - Von „--“ bis ⟨Zeilenende⟩

    Unterstützt in SQL-92, PostgreSQL, Oracle, SQL Server, IBM DB2, MySQL. MySQL benötigt ein Leerzeichen nach „--“, SQL-92 nicht. Access unterstützt diesen Kommentar nicht und auch nicht `/* ...*/`.
  - Von „/*“ bis „*/“

    Nur in PostgreSQL, Oracle, SQL Server und MySQL unterstützt: weniger portabel.

Formatfreie Sprache

- Obige Regel (beliebigere Leerplatz zwischen Token) bedeutet, dass SQL eine formatfreie Sprache wie z.B. Java ist:
  - Es ist nicht nötig, dass „SELECT“, „FROM“, „WHERE“ am Anfang neuer Zeilen stehen. Man kann auch die ganze Anfrage in eine Zeile schreiben.
  - Man kann z.B. komplexe Bedingungen auf mehrere Zeilen verteilen und Einrückungen verwenden, um die Struktur deutlich zu machen.

Bemerkung zu Syntaxdiagrammen (1)

- Eigentlich sind in der kontextfreien Grammatik, die die Syntax von SQL beschreibt, Konstanten und Bezeichner Terminalsymbole.
- Eigentlich nicht richtig:

  Tabelle:

- Eigentlich korrekt:

  Tabelle:

- Es gibt dann natürlich eine weitere Grammatik, die erklärt, wie Bezeichner aus einzelnen Zeichen aufgebaut sind.

Bemerkung zu Syntaxdiagrammen (2)

- Wir machen das hier nicht so, müssen aber doch deutlich erklären, was die lexikalischen Einheiten sind, zwischen denen Leerplatz, Kommentare, u.s.w. erlaubt sind.
  - Schlüsselworte (wie `WHERE`)
  - Bezeichner (wie `VORNAME`)
  - Konstanten (wie `101`)
  - Operatoren (wie `<=`), Klammern, Punkt
- Die Syntaxdiagramme definieren hier also nicht den Leerplatz zwischen Token.

  Teilweise ist Leerplatz zwischen Token sogar gefordert, z.B. zwischen einem Schlüsselwort und einem Bezeichner.

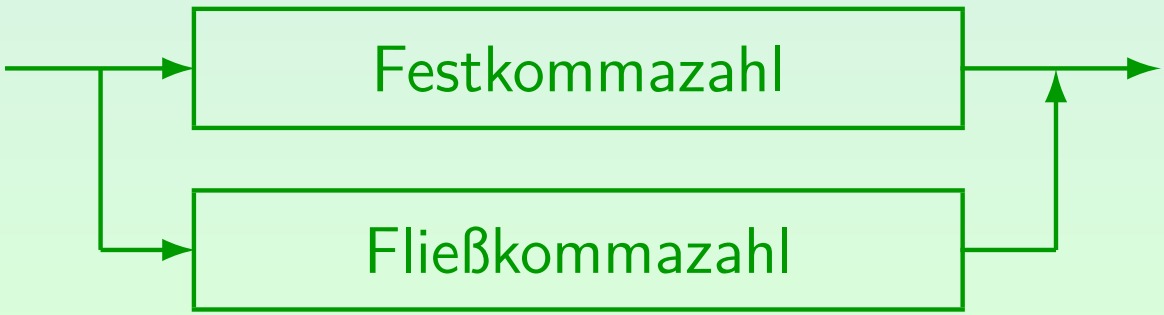
Inhalt

Zahlen (1)

- Numerische Literale sind Konstanten numerischer Datentypen (Fixpunkt- und Gleitkommazahlen).
- Z.B.: `1`, `+2.`, `-34.5`, `-.67E-8`
- Zahlen stehen nicht in Hochkommas!
- In SQL wird ein Dezimalpunkt verwendet, kein Komma!

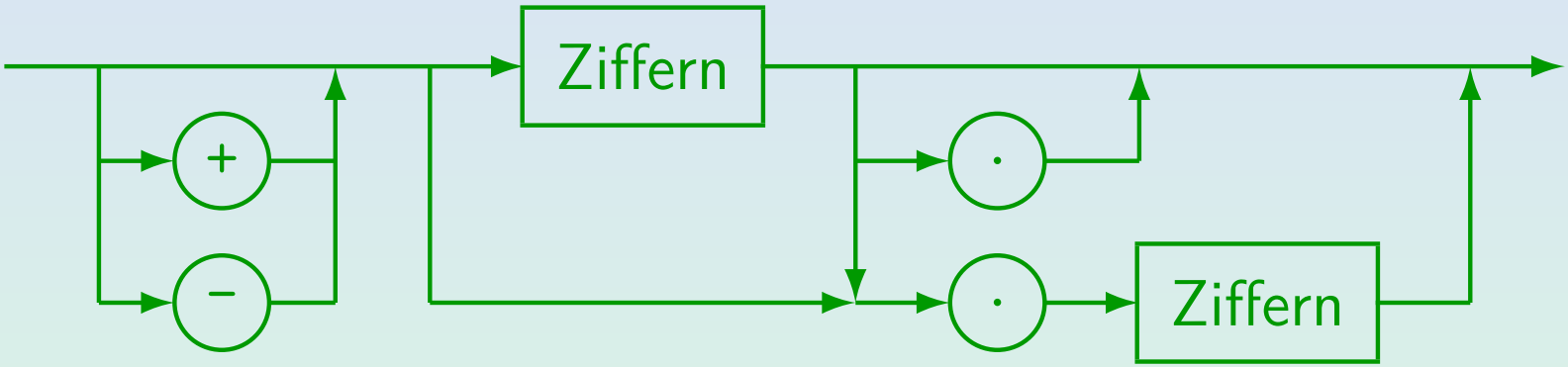
  Bei der Umwandlung von/nach Strings spielen eventuell Spracheinstellungen („Locale“) eine Rolle, ggf. auch bei Import/Export bzw. Ein-/Ausgaben. Nicht aber bei numerischen Literalen (Zahlkonstanten) direkt in SQL.

- Numerisches Literal:



Zahlen (2)

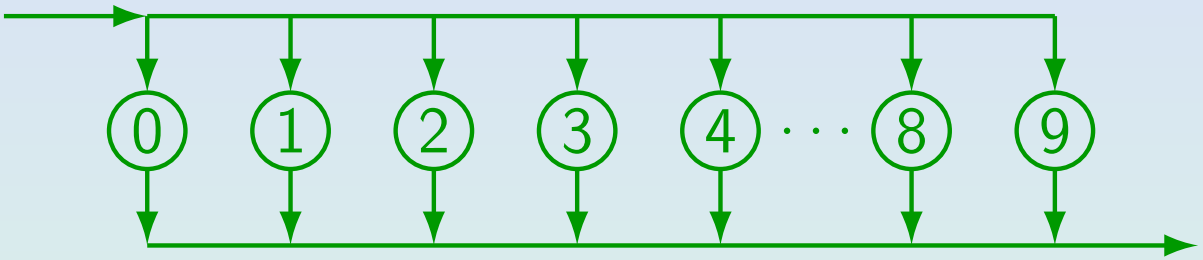
- Festkommazahl („Exact Numeric Literal“)



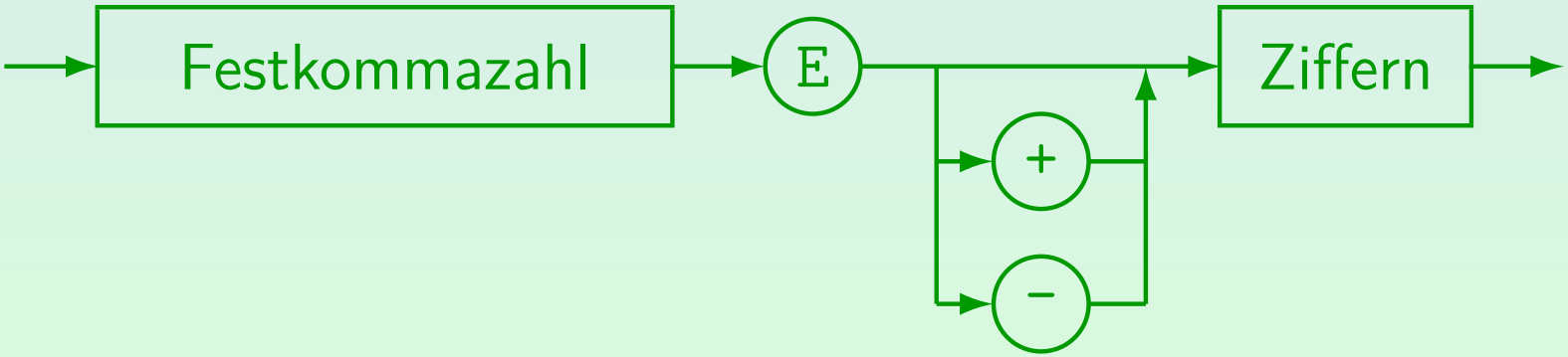
- Ziffern (vorzeichenlose ganze Zahl):

Zahlen (3)

- Ziffer:



- Fließkommazahl („Approximate Numeric Literal“):



Inhalt

Zeichenketten (1)

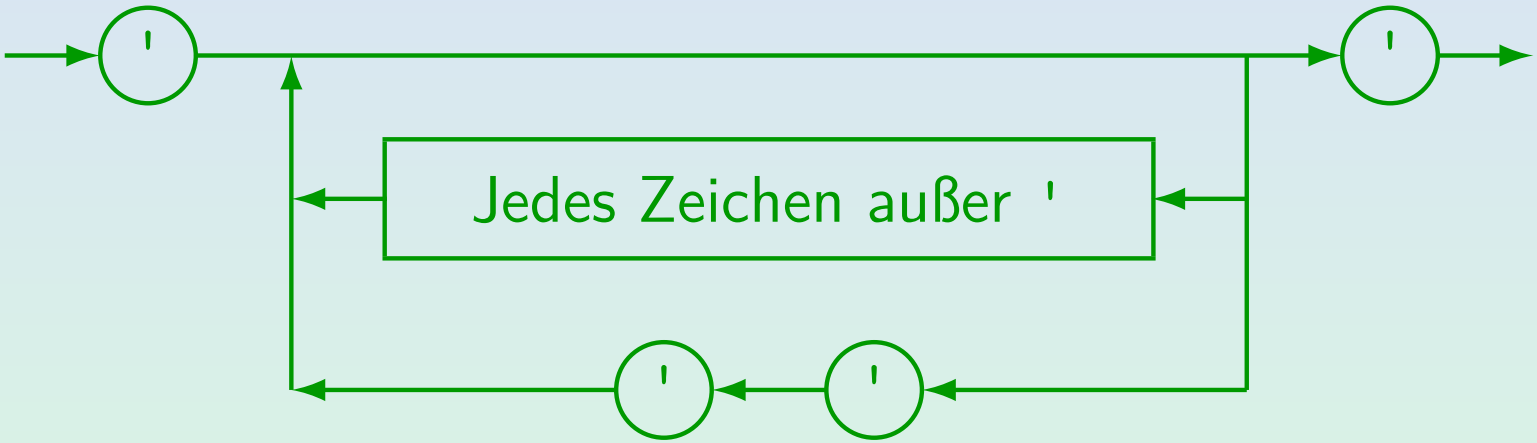
- Ein Zeichenketten-Literal ist eine Folge von Zeichen, eingeschlossen in Hochkommas, z.B.
  - `'abc'`
  - `'Dies ist ein String.'`
- Hochkommas in Zeichenketten müssen verdoppelt werden, z.B. `'John''s book'`.

  Der tatsächliche Wert der Zeichenkette ist `John's book` (mit einfachem Hochkomma). Das Verdoppeln ist nur eine Art, es einzugeben. Beachten Sie, dass die aus Java bekannten Escape-Sequenzen wie `\'` in SQL nicht vorgesehen sind (und z.B. in PostgreSQL auch tatsächlich nicht funktionieren).
- Natürlich ist auch die leere Zeichenkette erlaubt: `''`.

  In Oracle werden die leere Zeichenkette und der Nullwert identifiziert. Das entspricht nicht dem Standard.

Zeichenketten (2)

- String Literal:



Zeichenketten (3)

- SQL-92 erlaubt das Splitten von Zeichenketten (jedes Segment eingeschlossen in `'...'`) zwischen Zeilen.

  PostgreSQL und MySQL unterstützen diese Syntax, Oracle, SQL Server und Access nicht. Zeichenketten können aber mit dem Konkatenations-Operator (`||` in PostgreSQL und Oracle, `+` in SQL Server und Access) kombiniert werden.

- SQL-92 und alle sechs DBMS erlauben Zeilenumbrüche in Zeichenketten-Konstanten.

  D.h., das Hochkomma kann man auf einer folgenden Zeile schließen.

- Access und MySQL erlauben auch in Anführungszeichen `"` eingeschlossene String-Literale. Nicht konform zum Standard!

  Z.B. bei PostgreSQL und Oracle ist dies ein Syntaxfehler.

  Microsoft SQL Server hat die Option „`SET QUOTED_IDENTIFIER ON`“ (inzwischen standardmäßig gesetzt), die das standard-konforme Verhalten liefert.

Zeichenketten (4)

- Aus Programmiersprachen wie Java kennen Sie vielleicht „Escape-Sequenzen“ wie `\n` in Zeichenketten.

  So codiert man dort einen Zeilenumbruch.

- In SQL hat der Rückwärtsschrägstrich „\“ in Zeichenketten keine besondere Bedeutung, d.h. diese Escape-Sequenzen funktionieren nicht.

  Wenn man `'\n'` schreibt, ist das eine Zeichenkette aus zwei Zeichen, dem Rückwärts-Schrägstrich „\“ und dem Buchstaben „n“.

- MySQL interpretiert dagegen Escape-Sequenzen.

  Das verletzt den Standard. Es gibt Option `NO_BACKSLASH_ESCAPES`. [Doku]

- PostgreSQL hat eine Erweiterung gegenüber dem Standard und interpretiert Escape-Sequenzen, wenn man ein „`E`“ („escape string constant“) voranstellt: `E'\n'`. [Doku]

Inhalt

Andere Konstanten (1)

- Es gibt mehr Datentypen als nur Zahlen und Zeichenketten, z.B. (siehe Kapitel 5):
  - Zeichenketten mit nationalem Zeichensatz
  - Datum, Zeit, Zeitstempel, Datum-/Zeit-Intervall
  - Bit-Strings, binäre Daten
  - Large Objects (Dateien als Tabelleneintrag)
- Die Syntax der Konstanten dieser Datentypen ist allgemein sehr systemabhängig.

  Oft gibt es keine Konstanten dieser Typen, aber es gibt eine automatische Typ-Konvertierung („coercion“) von Strings.

Andere Konstanten (2)

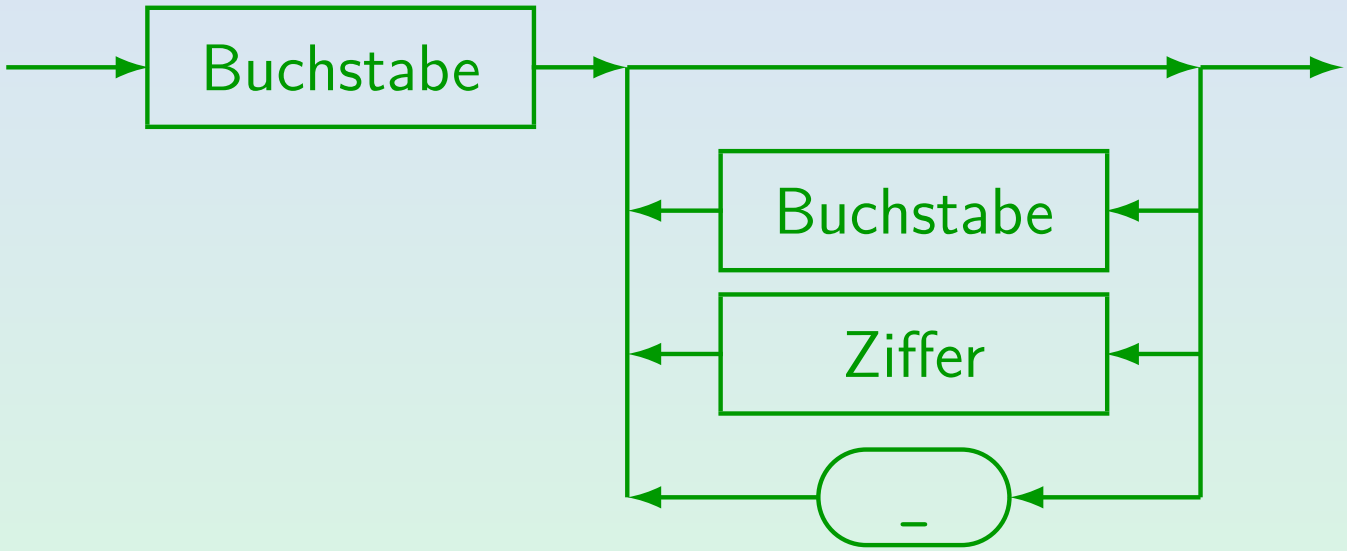
- Z.B. werden Datumswerte wie folgt geschrieben:
  - SQL-92-Syntax: `DATE '2002-10-31'` (also `YYYY-MM-DD`).
  - PostgreSQL und MySQL verstehen diese Syntax (auch ohne „`DATE`“), zusätzlich noch weitere Varianten.
  - Oracle: `'31-OCT-02'` (US), `'31.10.2002'` (D).

    Das Default-Format (Teil der nationalen Sprach-Einstellungen) wird automatisch konvertiert, ansonsten:
    `TO_DATE('31.10.2002','DD.MM.YYYY')`.
  - DB2: `'2002-10-31'`, `'10/31/2002'`, `'31.10.2002'`.
  - SQL Server: z.B. `'20021031'`, `'10/31/2002'`, `'October 31, 2002'` (abhängig von Sprache).
  - Access: `#10/31/2002#` (US), `#31.10.2002#` (D).

Inhalt

Bezeichner (1)

- Bezeichner:



- Z.B. `Dozenten_Name`, `X27`, aber nicht `_XYZ`, `12`, `2BE`.
- U.a. als Tabellen- und Spaltennamen verwendet.

Bezeichner (2)

- Bezeichner können bis 18 Zeichen haben (mind.).

System	Länge	Erstes Zeichen	Andere Zeichen
SQL-86	≤ 18	A-Z	A-Z,0-9
SQL-92	≤ 128	A-Z,a-z	A-Z,a-z,0-9,_
PostgreSQL	≤ 63	A-Z,a-z,_	A-Z,a-z,0-9,_,$
Oracle	≤ 30	A-Z,a-z	A-Z,a-z,0-9,_,#,$
MySQL	≤ 64	A-Z,a-z,0-9,_,$	A-Z,a-z,0-9,_,$
SQL Server	≤ 128	A-Z,a-z,_,(@,#)	A-Z,a-z,0-9,_,@,#,$
IBM DB2	≤ 18 (8)	A-Z,a-z	A-Z,a-z,0-9,_
Access	≤ 64	A-Z,a-z	A-Z,a-z,0-9,_

In MySQL müssen Bezeichner einen Buchstaben enthalten, ggf. aber nicht vorn.

- Müssen verschieden von reservierten Wörtern sein.

Es gibt viele reservierte Wörter, siehe unten. Einbettungen in Programmiersprachen (PL/SQL, Visual Basic) fügen noch mehr hinzu.

Bezeichner (3)

- Es ist möglich, nationale Zeichen zu verwenden.

  Das ist implementierungsabhängig. Z.B. wählt man in Oracle bei der Installation einen DB-Zeichensatz. Alphanumerische Zeichen von diesem Zeichensatz können in Bezeichnern verwendet werden.

- Bezeichner/Schlüsselwörter nicht case-sensitiv.

  Das scheint das zu sein, was der SQL-92-Standard sagt (das Buch von Date/Darwen über den Standard stellt es klar so dar). In PostgreSQL sind alle Bezeichner nicht case-sensitiv (werden in Kleinbuchstaben konvertiert). Oracle konvertiert alle Zeichen außerhalb von Hochkommas in Großbuchstaben. In SQL Server kann Case-Sensitivität bei der Installation gewählt werden. In MySQL hängt die Case-Sensitivität der Tabellennamen von der Case-Sensitivität von Dateinamen im zugrundeliegenden Betriebssystem ab (Tabellen als Dateien gespeichert). Innerhalb einer Anfrage muss man in MySQL konsistent bleiben. Schlüsselwörter und Spaltennamen sind in MySQL jedoch nie case-sensitiv.

Bezeichner (4)

- Betrachten Sie z.B. die Anfrage

```
SELECT X.VORNAME, X.NACHNAME
FROM   STUDENTEN X
```

- Folgende Anfrage ist vollkommen äquivalent:

```
select X . Vorname ,
       x.NachNaMe
  From Studenten X
```

(Dies zeigt auch die Formatfreiheit von SQL.)

Achtung: Zeichenketten-Vergleiche sind normalerweise case-sensitive:

```
select x.nachname from studenten x where x.vorname = 'lisa'
```

wird keine Antwort liefern (obwohl es `'Lisa'` gibt).

Reservierte Wörter (1)

1 = SQL-2016
2 = PostgreSQL 12
2a (Funktion/Typ ok)
2b (nur F./T. verboten)
3 = Oracle 12.1
4 = SQL Server 2017

— A —
ABS[1]
ACCESS[3]
ACOS[1]
ADD[3,4]
ALL[1,2,3,4]
ALLOCATE[1]
ALTER[1,3,4]
ANALYSE[2]
ANALYZE[2]
AND[1,2,3,4]
ANY[1,2,3,4]
ARE[1]
ARRAY[1,2]
ARRAY_AGG[1]
ARRAY_MAX_CARDINALITY[1]
AS[1,2,3,4]
ASC[3,2,4]
ASENSITIVE[1]
ASIN[1]
ASYMMETRIC[1,2]
AT[1]
ATAN[1]
ATOMIC[1]
AUTHORIZATION[1,2a,4]
AUDIT[3]
AVG[1]
— B —
BACKUP[4]
BEGIN[1,4]
BEGIN_FRAME[1]
BEGIN_PARTITION[1]
BETWEEN[1,2b,3,4]
BIGINT[1,2b]
BINARY[1,2a]
BIT[2b]
BLOB[1]
BOOLEAN[1,2b]
BOTH[1,2]
BREAK[4]
BROWSE[4]
BULK[4]
BY[1,3,4]
— C —
CALL[1]
CALLED[1]
CARDINALITY[1]
CASCADE[4]
CASCADED[1]
CASE[1,2,4]
CAST[1,2]
CEIL[1]
CEILING[1]
CHAR[1,2b,3]
CHAR_LENGTH[1]
CHARACTER[1,2b]
CHARACTER_LENGTH[1]
CHECK[1,2,3,4]
CHECKPOINT[4]
CLASSIFIER[1]
CLOB[1]
CLOSE[1,4]
CLUSTER[3]
CLUSTERED[4]
COALESCE[1,2b,4]
COLLATE[1,2,4]
COLLATION[2a]
COLLECT[1]
COLUMN[1,2,3,4]
COLUMN_VALUE[3]
COMMENT[3]
COMMIT[1,4]
COMPRESS[3]
COMPUTE[4]
CONCURRENTLY[2a]
CONDITION[1]
CONNECT[1,3]
CONSTRAINT[1,2,4]
CONTAINS[1,4]
CONTAINSTABLE[4]
CONTINUE[4]

Reservierte Wörter (2)

CONVERT[1,4]
COPY[1]
CORR[1]
CORRESPONDING[1]
COS[1]
COSH[1]
COUNT[1]
COVAR_POP[1]
COVAR_SAMP[1]
CREATE[1,2,3,4]
CROSS[1,2a,4]
CUBE[1]
CUME_DIST[1]
CURRENT[1,3,4]
CURRENT_CATALOG[1,2]
CURRENT_DATE[1,2,4]
CURRENT_DEFAULT_...
TRANSFORM_GROUP[1]
CURRENT_PATH[1]
CURRENT_ROLE[1,2]
CURRENT_ROW[1]
CURRENT_SCHEMA[1,2a]
CURRENT_TIME[1,2,4]
CURRENT_TIMESTAMP[1,2,4]
CURRENT_TRANSFORM_...
GROUP_FOR_TYPE[1]
CURRENT_USER[1,2,4]
CURSOR[1,4]
CYCLE[1]
DATABASE[4]
DATE[1,3]

— **D** —

DAY[1]
DBCC[4]
DEALLOCATE[1,4]
DEC[1,2b]
DECIMAL[1,2b,3]
DECFLOAT[1]
DECLARE[1,4]
DEFAULT[1,2,3,4]
DEFERRABLE[2]
DEFINE[1]
DELETE[1,3,4]
DENSE_RANK[1]
DENY[4]
DEREF[1]
DESC[2,4,3]
DESCRIBE[1]
DETERMINISTIC[1]
DISCONNECT[1]
DISK[4]
DISTINCT[1,2,3,4]
DISTRIBUTED[4]
DO[2]
DOUBLE[1,4]
DROP[1,3,4]
DUMP[4]
DYNAMIC[1]

— **E** —

EACH[1]
ELEMENT[1]
ELSE[1,2,3,4]
EMPTY[1]
END[1,2,4]
END-EXEC[1]
END_FRAME[1]
END_PARTITION[1]
EQUALS[1]
ERRLVL[4]
ESCAPE[1,4]
EVERY[1]
EXCEPT[1,2,4]
EXCLUSIVE[3]
EXEC[1,4]
EXECUTE[1,4]
EXISTS[1,2b,3,4]
EXIT[4]
EXP[1]
EXTERNAL[1,4]
EXTRACT[1,2b]

— **F** —

FALSE[1,2]
FETCH[1,2,4]
FILE[3,4]
FILLFACTOR[4]
FILTER[1]
FIRST_VALUE[1]
FLOAT[1,2b,3]

Reservierte Wörter (3)

FLOOR[1]
FOR[1,2,3,4]
FOREIGN[1,2,4]
FRAME_ROW[1]
FREE[1]
FREETEXT[4]
FREETEXTTABLE[4]
FREEZE[2a]
FROM[1,2,3,4]
FULL[1,2a,4]
FUNCTION[1,4]
FUSION[1]

— G —

GET[1]
GLOBAL[1]
GOTO[4]
GRANT[1,2,3,4]
GREATEST[2b]
GROUP[1,2,3,4]
GROUPING[1,2b]
GROUPS[1]

— H —

HAVING[1,2,3,4]
HOLD[1]
HOLDLOCK[4]
HOUR[1]

— I —

IDENTITY[1,4]
IDENTITY_INSERT[4]
IDENTITYCOL[4]
IDENTIFIED[3]
IF[4]
ILIKE[2a]
IMMEDIATE[3]
IN[1,2,3,4]
INCREMENT[3]
INDEX[3,4]
INDICATOR[1]
INITIAL[1,3]
INITIALLY[2]
INNER[1,2a,4]
INOUT[1,2b]
INSENSITIVE[1]
INSERT[1,3,4]
INT[1,2b]
INTEGER[1,2b,3]
INTERSECT[1,2,3,4]
INTERSECTION[1]
INTERVAL[1,2b]
INTO[1,2,3,4]
IS[1,2a,3,4]
ISNULL[2a]

— J —

JOIN[1,2a,4]
JSON_ARRAY[1]
JSON_ARRAYAGG[1]
JSON_EXISTS[1]
JSON_OBJECT[1]
JSON_OBJECTAGG[1]
JSON_QUERY[1]
JSON_TABLE[1]
JSON_TABLE_PRIMITIVE[1]
JSON_VALUE[1]

— K —

KEY[4]
KILL[4]

— L —

LAG[1]
LANGUAGE[1]
LARGE[1]
LAST_VALUE[1]
LATERAL[1,2]
LEAD[1]
LEADING[1,2]
LEAST[2b]
LEFT[1,2a,4]
LEVEL[3]
LIKE[1,2a,3,4]
LIKE_REGEX[1]
LIMIT[2]
LINENO[4]
LISTAGG[1]
LN[1]
LOAD[4]
LOCAL[1]
LOCALTIME[1,2]
LOCALTIMESTAMP[1,2]
LOCK[3]

Reservierte Wörter (4)

LOG[1]
LOG10[1]
LONG[3]
LOWER[1]
— **M** —
MATCH[1]
MATCH_NUMBER[1]
MATCH_RECOGNIZE[1]
MATCHES[1]
MAX[1]
MAXEXTENTS[3]
MEMBER[1]
MERGE[1,4]
METHOD[1]
MIN[1]
MINUS[3]
MINUTE[1]
MLSLABEL[3]
MOD[1]
MODE[3]
MODIFIES[1]
MODIFY[3]
MODULE[1]
MONTH[1]
MULTISET[1]
— **N** —
NATIONAL[1,2b,4]
NATURAL[1,2a]
NCHAR[1,2b]
NCLOB[1]
NESTED_TABLE_ID[3]
NEW[1]
NO[1]
NOAUDIT[3]
NOCHECK[4]
NOCOMPRESS[3]
NONCLUSTERED[4]
NONE[1,2b]
NORMALIZE[1]
NOT[1,2,3,4]
NOTNULL[2a]
NOWAIT[3]
NTH_VALUE[1]
NTILE[1]
NULL[1,3,4]
NULLIF[1,2b,4]
NUMBER[3]
NUMERIC[1]
— **O** —
OCTET_LENGTH[1]
OCCURRENCES_REGEX[1]
OF[1,3,4]
OFF[4]
OFFLINE[3]
OFFSET[1,2]
OFFSETS[4]
OLD[1]
OMIT[1]
ON[1,2,3,4]
ONE[1]
ONLINE[3]
ONLY[1,2]
OPEN[1,4]
OPENDATASOURCE[4]
OPENQUERY[4]
OPENROWSET[4]
OPENXML[4]
OPTION[3,4]
OR[1,2,3,4]
ORDER[1,2,3,4]
OUT[1,2b]
OUTER[1,2a,4]
OVER[1,4]
OVERLAPS[1,2a]
OVERLAY[1,2b]
— **P** —
PARAMETER[1]
PARTITION[1]
PATTERN[1]
PCTFREE[3]
PER[1]
PERCENT[1,4]
PERCENT_RANK[1]
PERCENTILE_CONT[1]
PERCENTILE_DISC[1]
PERIOD[1]
PERMANENT[4]
PIVOT[4]

Reservierte Wörter (5)

PLACING[2]
PLAN[4]
PORTION[1]
POSITION[1,2b]
POSITION_REGEX[1]
POWER[1]
PRECEDES[1]
PRECISION[1,2b,4]
PREPARE[1]
PRIMARY[1,2,4]
PRINT[4]
PRIOR[3]
PROC[4]
PROCEDURE[1,4]
PTF[1]
PUBLIC[3,4]

— **R** —

RAISERROR[4]
RANGE[1]
RANK[1]
RAW[3]
READ[4]
READS[1]
READTEXT[4]
REAL[1,2b]
RECONFIGURE[4]
RECURSIVE[1]
REF[1]
REFERENCES[1,2,4]
REFERENCING[1]
REGR_AVGX[1]
REGR_AVGY[1]
REGR_COUNT[1]
REGR_INTERCEPT[1]
REGR_R2[1]
REGR_SLOPE[1]
REGR_SXX[1]
REGR_SXY[1]
REGR_SYY[1]
RELEASE[1]
RENAME[3]
REPLICATION[4]
RESOURCE[3]
RESTORE[4]
RESTRICT[4]
RESULT[1]
RETURN[1,4]
RETURNING[2]
RETURNS[1]
REVERT[4]
REVOKE[1,3,4]
RIGHT[1,2a,4]
ROLLBACK[1,4]
ROLLUP[1]
ROW[1,2b,3]
ROW_NUMBER[1]
ROWCOUNT[4]
ROWGUIDCOL[4]
ROWID[3]
ROWNUM[3]
ROWS[1,3]
RULE[4]
RUNNING[1]

— **S** —

SAVE[4]
SAVEPOINT[1]
SCHEMA[4]
SCOPE[1]
SCROLL[1]
SEARCH[1]
SECOND[1]
SECURITYAUDIT[4]
SEEK[1]
SELECT[1,2,3,4]
SEMANTIC... KEYPHRASETABLE[4]
SEMANTIC... SIMILARITY...
SEMANTIC... DETAILSTABLE[4]
SEMANTIC... SIMILARITYTABLE[4]
SENSITIVE[1]
SESSION[3]
SESSION_USER[1,2,4]
SET[1,3,4]
SETOF[2b]
SETUSER[4]
SHARE[3]

Reservierte Wörter (6)

SHOW[1]
SHUTDOWN[4]
SIMILAR[1,2a]
SIN[1]
SINH[1]
SIZE[3]
SKIP[1]
SMALLINT[1,2b,3]
SOME[1,2,4]
SPECIFIC[1]
SPECIFICTYPE[1]
SQL[1]
SQLEXCEPTION[1]
SQLSTATE[1]
SQLWARNING[1]
SQRT[1]
START[1,3]
STATIC[1]
STATISTICS[4]
STDDEV_POP[1]
STDDEV_SAMP[1]
SUBMULTISET[1]
SUBSET[1]
SUBSTRING[1,2b]
SUBSTRING_REGEX[1]
SUCCEEDS[1]
SUCCESSFUL[3]
SUM[1]
SYMMETRIC[1,2]
SYNONYM[3]
SYSDATE[3]
SYSTEM[1]
SYSTEM_TIME[1]
SYSTEM_USER[1,4]

— **T** —

TABLE[1,2,3,4]
TABLESAMPLE[1,2a,4]
TAN[1]
TANH[1]
TEXTSIZE[4]
THEN[1,2,3,4]
TIME[1,2b]
TIMESTAMP[1,2b]
TIMEZONE_HOUR[1]
TIMEZONE_MINUTE[1]
TO[1,2,3,4]
TOP[4]
TRAILING[1,2]
TRAN[4]
TRANSACTION[4]
TRANSLATE[1]
TRANSLATE_REGEX[1]
TRANSLATION[1]
TREAT[1,2b]
TRIGGER[1,3,4]
TRIM[1,2b]
TRIM_ARRAY[1]
TRUE[1,2]
TRUNCATE[1,4]
TRY_CONVERT[4]
TSEQUAL[4]

— **U** —

UESCAPE[1]
UID[3]
UNION[1,2,3,4]
UNIQUE[1,2,3,4]
UNKNOWN[1]
UNNEST[1]
UNPIVOT[4]
UPDATE[1,3,4]
UPDATETEXT[4]
UPPER[1]
USE[4]
USER[1,2,3,4]
USING[1,2]

— **V** —

VALIDATE[3]
VALUE[1]
VALUES[1,2b,3,4]
VALUE_OF[1]
VAR_POP[1]
VAR_SAMP[1]
VARBINARY[1]
VARCHAR[1,2b,3]
VARCHAR2[3]
VARIADIC[2]
VARYING[1,4]
VERBOSE[2a]

Reservierte Wörter (7)

VERSIONING[1]
VIEW[3,4]
— **W** —
WAITFOR[4]
WHEN[1,2,4]
WHENEVER[1,3]
WHERE[1,2,3,4]
WHILE[4]
WIDTH_BUCKET[1]
WINDOW[1,2]
WITH[1,2,3,4]
WITHIN[1]
WITHIN GROUP[4]
WITHOUT[1]
WRITETEXT[4]
— **X** —
XMLATTRIBUTES[2b]
XMLCONCAT[2b]
XMLELEMENT[2b]
XMLEXISTS[2b]
XMLFOREST[2b]
XMLNAMESPACES[2b]
XMLPARSE[2b]
XMLPI[2b]
XMLROOT[2b]
XMLSERIALIZE[2b]
XMLTABLE[2b]
— **Y** —
YEAR[1]
— **Z** —

- Diese Liste hat 526 Einträge.

  SQL-2016: 365, PostgreSQL: 151, Oracle: 111, MS SQL Server: 187.

  [https://www.postgresql.org/docs/12/sql-keywords-appendix.html]

  [https://docs.oracle.com/database/121/SQLRF/ap_keywd001.htm]

  [https://docs.microsoft.com/de-de/sql/t-sql/language-elements/reserved-keywords-transact-sql]

- Java hat 50 reservierte Worte. C hat 30.

Reservierte Wörter (8)

- Nicht alle Worte mit Spezialbedeutung sind reserviert.

  Manchmal ist syntaktisch aus dem Kontext klar, dass dort keine nutzerdefinierte Tabelle oder Spalte stehen kann. Im SQL Standard gibt es neben der Liste der reservierten Worte explizit noch eine Liste „non-reserved word“ von Worten, die in der Grammatik vorkommen, aber nicht reserviert sind.
  Bei PostgreSQL sind Worte teils nur für Typen und Funktionen reserviert.

- Es kommt vor, dass man einen Tabellen- oder Spaltennamen wählt, der ein reserviertes Wort ist.

  Dann bekommt man eventuell eine ganz merkwürdige Fehlermeldung.

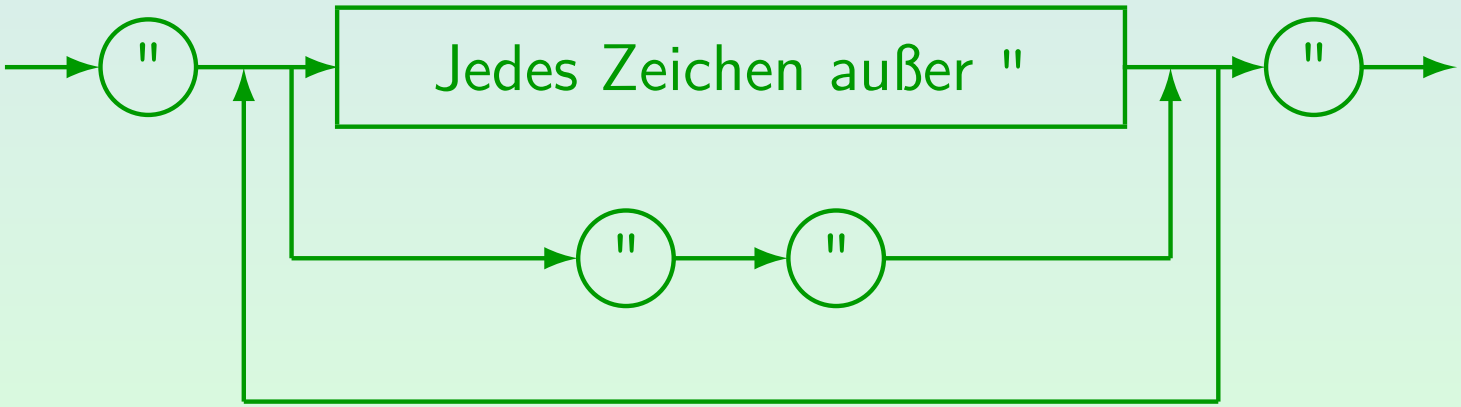
- Es gibt Seiten, um den Status eines Bezeichners in verschiedenen DBMS zu prüfen.

  Z.B. [https://www.petefreitag.com/tools/sql_reserved_words_checker/]
  Reservierte Worte sind auch ein Portabilitätsproblem: Die verschiedenen DBMS unterschieden sich nicht unwesentlich.

Delimited Identifier (1)

- Es ist möglich, jede Zeichenfolge in Anführungszeichen als Bezeichner zu verwenden, z.B. `"id, 2!"`.

  Solche Bezeichner sind case-sensitiv und es gibt keine Konflikte mit reservierten Wörtern. SQL-86 enthält dies nicht. In Deutsch würde „Delimited Identifier“ in etwa „Abgegrenzte Bezeichner“ heißen.



Delimited Identifier (2)

- Delimited Identifier sind keine String-Konstanten!
  String-Konstanten haben die Form `'...'` (mit Hochkommas).

  SQL Server akzeptiert ’ und " für String-Konstanten und nimmt `[...]` für Delimited Identifier. „`SET QUOTED_IDENTIFIER ON`“ schaltet auf den SQL-92-Standard um (aber Delimited Identifier sind nicht case-sensitive). Access versteht `[...]` und ‘...‘ für Delimited Identifier, schließt aber die Zeichen `!.‘[]"` und Leerzeichen am Anfang aus.

- Wenn man z.B. in PostgreSQL oder Oracle schreibt:

  ```
  SELECT * FROM STUDENTEN WHERE VORNAME = "Lisa"
  ```

  Fehler: `"Lisa"` ist ein ungültiger Spaltenname.

  Delimited Identifier werden normalerweise nur verwendet, um ausgegebene Spaltennamen umzubenennen (oder wenn Spaltennamen in einer neuen DBMS-Version zu reservierten Wörtern werden).

Delimited Identifier (3)

- Delimited Identifier werden hauptsächlich verwendet, um Ausgabe-Spalten umzubenennen, z.B.

```
SELECT VORNAME AS "Vorname", NACHNAME "Name"
FROM   STUDENTEN
```

  „AS“ ist optional (außer in MS Access).

  Man kann sich fragen, warum SQL „Delimited Identifier“ hat, aber z.B. Java nicht. Die Antwort ist, dass z.B. Variablennamen in Java-Programmen etwas rein Internes sind (nur für den Programmierer), aber zumindest Spaltennamen in SQL auch in der Ausgabe erscheinen (und deswegen nicht auf die normale Bezeichner-Syntax eingeschränkt sein sollten).

- Ist aber der neue Spaltenname ein legaler Bezeichner, sind die Anführungszeichen unnötig:

```
SELECT VORNAME AS V_NAME, NACHNAME Name
FROM   STUDENTEN
```

Delimited Identifier (4)

- In PostgreSQL werden normale Bezeichner (ohne "...") in Kleinbuchstaben ausgegeben.

  In Oracle in Großbuchstaben. Man sollte sich also nicht darauf verlassen, dass Bezeichner ohne "..." auf Kleinbuchstaben abgebildet werden. Wenn z.B. ein Spaltenname abc im `CREATE TABLE` ohne "..." angegeben wurde, funktioniert es in PostgreSQL zufällig, wenn man ihn später als "abc" anspricht. In Oracle würde es dagagen nicht funktionieren. Man sollte Spaltennamen immer gleich schreiben, zumindest nicht einmal als „Delimited Identifier“ und einmal als normaler Bezeichner.

- Wenn man SQL-Anfragen als Zeichenketten in Java-Programme einfügt (z.B. für die JDBC-Schnittstelle), sind „Delimited Identifier“ ungünstig, weil "..." auch als String-Begrenzer in Java verwendet werden.

  Obwohl möglich, sollten Tabellen und Spalten in der Datenbank also eher nicht als Delimited Identifier geschrieben werden.

Inhalt

Lexikalische Fehler

- Anführungszeichen, z.B. `"Lisa"`, für String-Literale verwenden (Delimited Identifier, kein String).

  Manche Systeme erlauben `"..."`, aber das verletzt den Standard.

- Hochkommas für Zahlen verwenden, z.B. `'123'`.

  Das sollte einen Typfehler geben. Das DBMS könnte jedoch einfach den Typ von einem der Operanden konvertieren. Da < usw. für Strings und Zahlen anders definiert ist, kann dies gefährlich sein und sollte vermieden werden. Z.B. `'12' < '3'`.

- Reservierte Wörter als Tabellen-, Spalten- oder Tupelvariablennamen verwenden.

  Die Fehlermeldung könnte seltsam sein (nicht verständlich). Daher sollte man diese Möglichkeit im Auge behalten.

Begrenzung von SQL-Anfragen

- In der Kommandozeilen-Schnittstelle `psql` von PostgreSQL (und ähnlichen Programmen) muss jedes SQL-Statement mit einem Semikolon „;“ abgeschlossen werden

  Da SQL-Statements über mehrere Zeilen gehen können, ist dies notwendig, damit `psql` weiß, wann das SQL-Statement beendet ist. Auch wenn SQL in C-Programme eingebettet ist, wird das Semikolon als Begrenzer verwendet. Auch mehrere Anfragen hintereinander im Adminer-Eingabefenster müssen durch „;“ getrennt werden (bei einer Anfrage ist das „;“ dagegen unnötig).

- Aber eigentlich gehört das Semikolon nicht zum SQL-Statement.

  Beim Adminer-Webinterface u.a. für PostgreSQL ist kein Semikolon erforderlich (man schreibt die Anfrage in das Eingabefeld und drückt den „`Execute`“-Knopf, um das Ende zu signalisieren). Ein Semikolon könnte sogar ein Fehler sein, wie im Kommandozeilen-Interface von DB2. Auch wenn SQL-Anweisungen in String-Konstanten geschrieben werden (z.B. JDBC), ist kein Semikolon nötig.