Teil 10: Tabellendefinition

Literatur:

- Elmasri/Navathe:Fundamentals of Database Systems, 3. Auflage, 1999. Kap. 8, "SQL — The Relational Database Standard"
- Kemper/Eickler: Datenbanksysteme, 4. Auflage, Oldenbourg, 1997. Kapitel 4: Relationale Anfragesprachen.
- Lipeck: Skript zur Vorlesung Datenbanksysteme, Univ. Hannover, 1996.
- Date/Darwen: A Guide to the SQL Standard, 4. Auflage, Addison-Wesley, 1997.
- van der Lans: SQL, Der ISO-Standard, Hanser, 1990.
- Melton/Simon: Understanding the New SQL. Morgan Kaufman, 1993.
- Oracle 8i SQL Reference, Release 2 (8.1.6), Dez. 1999, Part No. A76989-01.
- Oracle 8i Concepts, Release 2 (8.1.6), Dez. 1999, Part No. 76965-01. Kapitel 12: Built-in Datatypes.
- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data.
- Microsoft Jet Database Engine Programmer's Guide, 2. Auflage (Teil der MSDN Library Visual Studio 6.0). Microsoft Access 2000 Online-Hilfe.
- DuBois: MySQL. New Riders Publishing, 2000, ISBN 0-7357-0921-1, 756 Seiten.
- MySQL-Handbuch für Version 3.23.53.

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- `CREATE TABLE`-Anweisungen in SQL schreiben.
- Integritätsbedingungen in SQL definieren.

  `NOT NULL`, Schlüssel, Fremdschlüssel und `CHECK`.
- für ein Attribut einen Datentyp wählen.

  Sie sollten die üblichen Datentypen, die es in jedem DBMS geben sollte, aufzählen und erklären können. Sie sollten die Parameter von `NUMERIC`, `CHAR` und `VARCHAR` erklären können. Sie sollten auch wissen, welche weiteren Datentypen es noch geben könnte (für die Einzelheiten können Sie dann in der Anleitung Ihres DBMS nachschauen).
- einige Datentyp-Funktionen aufzählen (Beispiele).

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).

  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 Datentypfunktionen +, −, *, /.

  Die Existenz dieser Funktionen kann man in jedem DBMS erwarten.
- Andere Funktionen sind von DBMS zu DBMS sehr unterschiedlich.

  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:
  - ◇ Lange Zeichenketten
  - ◇ Binäre Daten
  - ◇ Zeichenketten in nationalem Zeichensatz
  - ◇ Datums- und Zeitwerte
- Benutzer-definierte und DBMS-spezifische Typen.

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 Datentyp `CHAR`(n) war bereits im SQL-86-Standard enthalten.

  Natürlich wird er von allen fünf Systemen unterstützt (Oracle, SQL Server, DB2, Access und MySQL).
- Die Systeme unterscheiden sich im maximalen Wert für die Länge n.

DBMS	Maximales n
Oracle 8.0	2000
DB2 UDB 5	254
SQL Server 7	8000
Access	255
MySQL	255

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.

- 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 fünf in dieser Vorlesung besprochenen DBMS unterstützt (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 8.0	4000
DB2 UDB 5	254/4000
SQL Server 7	8000
Access	255
MySQL	255

Ist in DB2 n größer als 254, ist für diese Spalte keine Sortierung möglich (einschließlich `ORDER BY`, `GROUP BY`, `DISTINCT`).

Zeichenketten-Funktionen (1)

Zeichenketten-Funktionen in Oracle:

- s_1`||`s_2, `CONCAT`(s_1, s_2).
- `LENGTH`(s), `INSTR`(s, x), `INSTR`(s, x, n, m).
- `INITCAP`(s), `LOWER`(s), `UPPER`(s), `TRANSLATE`(s, x, y).
- `LPAD`(s, n), `LPAD`(s, n, p), `LTRIM`(s), `LTRIM`(s, x), `RPAD`(s, n), `RPAD`(s, n, p), `RTRIM`(s), `RTRIM`(s, x).
- `SUBSTR`(s, m), `SUBSTR`(s, m, n), `REPLACE`(s, x, y).
- `ASCII`(s), `CHR`(n).
- `SOUNDEX`(s).

Zeichenketten-Funktionen (2)

Zeichenketten-Funktionen im SQL-92-Standard:

- s_1`||`s_2.
- `CHARACTER_LENGTH`(s), `OCTET_LENGTH`(s).

  `CHARACTER_LENGTH`(s) kann mit `CHAR_LENGTH`(s) abgekürzt werden.

- `LOWER`(s), `UPPER`(s).
- `POSITION`(s_1 `IN` s_2).
- `SUBSTRING`(s `FROM` m `FOR` n).
- `TRIM`(s), `TRIM(LEADING` c `FROM` s), `TRIM(TRAILING` c `FROM` s), `TRIM(BOTH` c `FROM` s).

Zeichenketten-Funktionen (3)

Zeichenketten-Funktionen in DB2:

- s_1||s_2, CONCAT(s_1, s_2).
- LENGTH(s).
- LOCATE(s_1, s_2), LOCATE(s_1, s_2, n), POSSTR(s_1, s_2).
- LTRIM(s), RTRIM(s).
- INSERT(s_1, n, l, s_2), REPLACE(s, f, t), LEFT(s, n), RIGHT(s, n), SUBSTR(s, m), SUBSTR(s, m, n).
- LCASE(s), UCASE(s), TRANSLATE(s), TRANSLATE(s, t, f), TRANSLATE(s, t, f, p).

Zeichenketten-Funktionen (4)

Zeichenketten-Funktionen in DB2, fortgesetzt:

- ASCII(s), CHR(n).
- DIFFERENCE(s_1, s_2), SOUNDEX(s).
- GENERATE_UNIQUE(), REPEAT(s, n), SPACE(n).

Zeichenketten-Funktionen (5)

Zeichenketten-Funktionen in SQL Server:

- s_1+s_2 (Konkatenation).
- LEN(s).
- ASCII(s), CHAR(n), UNICODE(s), NCHAR(n).
- LOWER(s), UPPER(s).
- LTRIM(s), RTRIM(s).
- LEFT(s, n), RIGHT(s, n), SUBSTRING(s, m, n).
- CHARINDEX(s_1, s_2), CHARINDEX(s_1, s_2, n), PATINDEX(s_1, s_2).

Zeichenketten-Funktionen (6)

Zeichenketten-Funktionen in SQL Server, fortgesetzt:

- REPLACE(s, x, y), STUFF(s, n, m, x).

  REPLACE ersetzt jedes Auftreten von x in s durch y. STUFF ersetzt Zeichen mit Position n bis $n + m$ in s durch x.
- DIFFERENCE(s_1, s_2), SOUNDEX(s).
- QUOTENAME(s), QUOTENAME(s, q).
- REPLICATE(s, n), SPACE(n).
- REVERSE(s).
- STR(x), STR(x, l), STR(x, l, d).

  Konvertiert eine Zahl in einen String. l ist die Output-Länge.

Zeichenketten-Funktionen (7)

Zeichenketten-Funktionen in MySQL:

- CONCAT($s_1, s_2, \ldots$), CONCAT_WS($s, s_1, s_2, \ldots$).
- LENGTH(s), OCTET_LENGTH(s), CHAR_LENGTH(s), CHARACTER_LENGTH(s).
- LOCATE(s_1, s_2), POSITION(s_1 IN s_2), LOCATE(s_1, s_2, n), INSTR(s, x).
- LCASE(s), LOWER(s), UCASE(s), UPPER(s).
- LPAD(s, n, p), LTRIM(s), RPAD(s, n, p), RTRIM(s), TRIM(s), TRIM(LEADING c FROM s), TRIM(TRAILING c FROM s), TRIM(BOTH c FROM s).

Zeichenketten-Funktionen (8)

Zeichenketten-Funktionen in MySQL, fortgesetzt:

- LEFT(s, n), RIGHT(s, n), SUBSTRING(s, m, n), SUBSTRING(s FROM m FOR n), MID(s, n), SUBSTRING(s, m), SUBSTRING(s FROM m), SUBSTRING_INDEX(s, d, n).
- INSERT(s_1, n, m, s_2).
- ASCII(s), ORD(s), CHAR($n_1, \ldots$).
- SOUNDEX(s).
- SPACE(n), REPEAT(s, n).

Zeichenketten-Funktionen (9)

Zeichenketten-Funktionen in MySQL, fortgesetzt:

- REVERSE(s).
- CONV(s, b_1, b_2), CONV(n, b_1, b_2), BIN(n), OCT(n), HEX(n).
- ELT($n, s_1, s_2, \ldots$), FIELD($s, s_1, s_2, \ldots$).
- FIND_IN_SET(s_1, s_2), MAKE_SET($n, s_1, s_2, \ldots$), EXPORT_SET(n, s_1, s_2, s_3, m).
- LOAD_FILE(f).

Zeichenketten-Funktionen(10)

Zeichenketten-Funktionen in Access:

- ASC(s), ASCB(s), ASCW(s), CHR(n), CHRB(n), CHRW(n).
- CHOOSE($n, s_1, s_2, \ldots$).
- CURDIR(), CURDIR(c), DIR(p), DIR(p, a).
- ENVIRON(v), ENVIRON(n).
- ERROR(), ERROR(n).
- FORMAT(x), FORMAT($x, f, \ldots$), FORMATCURRENCY($x, \ldots$), FORMATDATETIME($x, \ldots$), FORMATNUMBER($x, \ldots$), FORMATPERCENT($x, \ldots$).

Zeichenketten-Funktionen(11)

Zeichenketten-Funktionen in Access, fortgesetzt:

- HEX(n), OCT(n), STR(n), CSTR(n).
- INSTR(s_1, s_2), INSTR(n, s_1, s_2), INSTR(n, s_1, s_2, c), INSTRB($\ldots$), INSTRREV($s_1, s_2, \ldots$), REPLACE($s, s_1, s_2, \ldots$).
- LCASE(s), UCASE(s), STRCONV(s, c, l).
- LEFT(s, n), LEFTB(s, n), MID(s, n), MID(s, n, m), RIGHT(s, n), RIGHTB(s, n).
- LEN(s), LENB(s).

Zeichenketten-Funktionen(12)

Zeichenketten-Funktionen in Access, fortgesetzt:

- LTRIM(s), RTRIM(s), TRIM(s).
- MONTHNAME(n), MONTHNAME(n, b). WEEKDAYNAME($n, \ldots$).
- SPACE(n), STRING(n, c).
- STRCOMP(s_1, s_2), STRCOMP(s_1, s_2, c).
- STRREVERSE(s).
- TYPENAME(x).

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.

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.

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)

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).

Zahlen (4)

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

DB2, SQL Server, MySQL und Access verwenden 32 Bit-Binärzahlen: -2147483648 (-2^{31}) .. +2147483647 $(2^{31}-1)$. 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.

DB2, SQL Server, MySQL und Access verwenden 16 Bit-Binärzahlen: -32768 (-2^{15}) .. +32767 $(2^{15}-1)$. Somit ist der Bereich in diesen Systemen größer als NUMERIC(4), aber kleiner als NUMERIC(5).
In Oracle wieder ein Synonym für NUMBER(38).

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.

Zahlen (6)

- 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).

Zahlen (7)

- 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.

Zahlen(8)

Operationen für Zahlen in Oracle:

- $x + y$, $x - y$, $x * y$, x / y, $-x$, $+x$.
- `ABS`(x), `SIGN`(x).
- `SIN`(x), `SINH`(x), `ASIN`(x), `COS`(x), `COSH`(x), `ACOS`(x), `TAN`(x), `TANH`(x), `ATAN`(x), `ATAN2`(x, y).
- `CEIL`(x), `FLOOR`(x), `ROUND`(x), `ROUND`(x, n), `TRUNC`(x, n).
- `EXP`(x), `LN`(x), `LOG`(b, x), `POWER`(x, y),
- `MOD`(m, n).
- `SQRT`(x).

Zahlen (9)

Operationen für Zahlen im SQL-92-Standard:

- $x + y$, $x - y$, $x * y$, x / y, $-x$, $+x$.

Operationen für Zahlen in DB2:

- $x + y$, $x - y$, $x * y$, x / y, $-x$, $+x$.
- `ABS`(x), `SIGN`(x).
- `SIN`(x), `ASIN`(x), `COS`(x), `ACOS`(x), `TAN`(x), `COT`(x), `ATAN`(x), `ATAN2`(x, y).
- `CEIL`(x), `FLOOR`(x), `ROUND`(x, n), `TRUNC`(x, n).

Zahlen (10)

Operationen für Zahlen in DB2, fortgesetzt:

- `EXP`(x), `LN`(x), `LOG10`(x), `POWER`(x, y).
- `MOD`(m, n).
- `SQRT`(x).
- `RAND`().
- `DEGREES`(x).

Zahlen (11)

Operationen für Zahlen in SQL Server:

- $x + y$, $x - y$, $x * y$, x / y, $x \% y$ (modulo), $-x$, $+x$.
- $x \& y$, $x | y$, $x \hat{} y$, $\tilde{} x$ (Bit-Operationen).
- ABS(x), SIGN(x).
- SIN(x), ASIN(x), COS(x), ACOS(x), TAN(x), ATAN(x), ATN2(x, y), COT(x).
- CEILING(x), FLOOR(x), ROUND(x, n), ROUND($x, n, 1$) (schneidet ab).

Zahlen (12)

Operationen für Zahlen in SQL Server, fortgesetzt:

- EXP(x), LOG(x), LOG10(x), POWER(x, y).
- SQRT(x), SQUARE(x).
- DEGREES(x), RADIANS(x), PI().
- RAND(), RAND(n).

Zahlen (13)

Operationen für Zahlen in MySQL:

- $x + y$, $x - y$, $x * y$, x / y, $-x$.
- ABS(x), SIGN(x).
- SIN(x), ASIN(x), COS(x), ACOS(x), TAN(x), ATAN(x), ATAN(x, y), ATAN2(x, y), COT(x).
- CEIL(x), FLOOR(x), ROUND(x), ROUND(x, n), TRUNCATE(x, n).
- EXP(x), LOG(x), LOG10(x), POW(x, y), POWER(x, y).
- MOD(m, n), $n \% m$.

Zahlen (14)

Operationen für Zahlen in MySQL, fortgesetzt:

- SQRT(x).
- PI(), DEGREES(x), RADIANS(x).
- RAND(), RAND(n).

  Es gibt Einschränkungen in der Verwendung von RAND.

Zahlen (15)

Operationen für Zahlen in Access:

- $x + y$, $x - y$, $x * y$, x / y, $-x$, $+x$.
- ABS(x), SGN(x).
- SIN(x), COS(x), TAN(x), ATN(x).
- ROUND(x), ROUND(x, n), FIX(x), INT(x).
- EXP(x), LOG(x).
- n MOD m.
- SQR(x).

Außerdem gibt es Funktionen, um Zinszahlungen zu berechnen etc.

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.

Vier Systeme (Oracle, DB2, SQL Server, MySQL) verstehen sie. Access unterstützt NUMERIC, DECIMAL, FLOAT(p), DOUBLE PRECISION nicht. Alle fünf Systeme unterstützen VARCHAR, was nicht im SQL-86-Standard enthalten war.

Inhalt

Lange Zeichenketten (1)

Oracle:

- LONG: Zeichenketten von bis zu 2GB Länge.
- Die Verwendung von LONG-Spalten ist sehr eingeschränkt. Im wesentlichen ist nur möglich, eine Datei in der DB zu speichern und sie wieder abzurufen, aber man kann sie nicht in Bedingungen oder Berechnungen in SQL verwenden.

Z.B. können LIKE, ||, LENGTH und andere Zeichenketten-Funktionen nicht für LONG-Werte verwendet werden. Eine Tabelle darf maximal eine Spalte vom Typ LONG haben. Die Eingabe/Ausgabe von LONG-Werten ist wie gewohnt möglich, z.B. mit SELECT. In SQL*Plus legt SET LONG n die maximale Ausgabelänge fest.

Lange Zeichenketten (2)

Oracle, fortgesetzt:

- Wird für einen langen Text `LIKE` benötigt, muss er in Zeilen/Abschnitte geteilt werden, die separat als `VARCHAR`-Werte gespeichert werden.
- `CLOB`: Character large object ("großes Zeichenobjekt"), bis zu 4GB.

  Das ist wie eine in der DB gespeicherte Datei mit einer eigenen Identität (LOB-Locator). Es ist `LONG` sehr ähnlich.
- `CLOB` ist neu in Oracle8. Oracle7 hatte nur `LONG`.

  Wahrscheinlich wird `LONG` nur für Abwärts-Kompatibilität unterstützt.

Lange Zeichenketten (3)

Oracle, fortgesetzt:

- Unterschiede zwischen `CLOB` und `LONG` sind z.B.:
  - ◇ Die Programmierschnittstelle erlaubt wahlfreien Zugriff auf die `CLOB`-Daten (beliebiger Teilstring).

    `LONG`-Werte können nur sequentiell gelesen werden.
  - ◇ Tabelle kann mehrere `CLOB`-Spalten enthalten.
  - ◇ `CLOB`-Werte können an anderen Orten gespeichert werden als die Tabelle, in der sie auftauchen.
- `CLOB`-Werte darf man nur über PL/SQL-Prozeduren (im Paket `DBMS_LOB`) in Bedingungen verwenden.

Lange Zeichenketten (4)

DB2:

- Auch DB2 hat character large objects (bis zu 2GB).

  Character large objects werden mit einer maximalen Größe versehen, z.B. `CLOB(1M)`, was die Länge des Deskriptors beeinflusst, der verwendet wird, um auf die eigentlichen Daten zu zeigen. Diese Daten werden separat von den Tabellenzeilen gespeichert.
- Schon `VARCHAR`-Spalten der Länge > 254 kann man nicht in `ORDER BY`, `GROUP BY`, `DISTINCT` verwenden.

  Alles, was Sortierung benötigt, ist ausgeschlossen.
- Außerdem können `CLOB`(n)-Spalten nicht mit `=`, `<>`, `<`, `<=`, `>`, `>=`, `IN`, `BETWEEN` verwendet werden.

Lange Zeichenketten (5)

DB2, fortgesetzt:

- `CLOB`(n)-Spalten können jedoch mit `LIKE` verwendet werden.
- Es gibt auch einen Datentyp `LONG VARCHAR` (bis zu 32700 Bytes), der wegen Abwärtskompatibilität beibehalten wurde.

Lange Zeichenketten (6)

SQL Server:

- SQL Server hat einen Datentyp "TEXT", der bis zu 2GB speichern kann.

  Eigentlich ist die maximale Größe $2^{31} - 1$, d.h. 2 147 483 647.

- TEXT-Spalten können in der WHERE-Klausel nur mit LIKE oder IS NULL verwendet werden.

  Z.B. können `=`, `<>`, `<`, `<=`, `>`, `>=`, IN, BETWEEN nicht verwendet werden.

- TEXT-Spalten können nicht mit DISTINCT, ORDER BY, GROUP BY verwendet werden.

Lange Zeichenketten (7)

SQL Server, fortgesetzt:

- TEXT-Spalten sind nicht als Argumente für + (String-Konkatenation) erlaubt, aber es gibt einige Datentyp-Funktionen wie DATALENGTH oder SUBSTRING, die mit TEXT funktionieren.

Lange Zeichenketten (8)

MySQL:

- BLOB: "Binary large object" von bis zu 64 KBytes.

  Für binäre Daten. Genaue maximale Größe: $2^{16} - 1 = 65\,535$ Bytes.

- TEXT: Zeichenkette von bis zu 64 KBytes.

  Der einzige Unterschied zwischen BLOB und TEXT ist, dass Vergleiche für BLOB case-sensitive und für TEXT case-insensitive sind.

- MEDIUMBLOB/MEDIUMTEXT: Max. 16 MB.

  Die genaue maximale Länge ist $2^{24} - 1 = 16\,777\,215$.

- LONGBLOB/LONGTEXT: Max. 4 GB.

  Die genaue maximale Länge ist theoretisch $2^{32} - 1 = 4\,294\,967\,295$. Die derzeitige Version von MySQL hat jedoch eine Grenze von 16 MB.

Lange Zeichenketten (9)

MySQL, fortgesetzt:

- Im Allgemeinen werden BLOB und TEXT wie VARCHAR(n) mit großem n behandelt.

  MySQL entfernt jedoch Leerzeichen am Ende, wenn VARCHAR-Werte gespeichert werden, tut dies aber nicht für BLOB und TEXT.

- Sortierung (GROUP BY, ORDER BY, DISTINCT) funktioniert nur für die ersten 1024 Bytes.

  Diese Grenze kann mit dem Parameter max_sort_length erhöht werden. Seit Version 3.23.2 kann man Indexe auf BLOB- und TEXT-Spalten haben.

- Empfehlung: SUBSTRING-Funktion verwenden, um einen Wert zu extrahieren, der sortiert werden kann.

Lange Zeichenketten (10)

Access:

- `MEMO`: Lange Textdaten.

  Die maximale Größe ist 65.535 Zeichen, wenn man die Daten über das Nutzer-Interface eingibt, und 1 GB, wenn man die Daten über das Programm-Interface eingibt. Sogar in den Handbüchern werden verschiedene Werte erwähnt: 64.000 Zeichen, 65.535 Zeichen, 1.2 GB und 2.14 GB. `LONGTEXT` ist ein Synonym für diesen Datentyp.

- `OLEOBJECT`: Z.B. Microsoft Word-Dokument.

  Maximale Größe ist 1 GB. `LONGBINARY` ist ein Synonym für diesen Typ. Man kann `DISTINCT`, `GROUP BY` oder `ORDER BY` für Werte dieses Typs nicht verwenden. Das Handbuch erwähnt die gleichen Einschränkungen für `MEMO`-Daten, aber es schien dort zu funktionieren (in Access 2000). `OLEOBJECT`-Werte werden als "Long binary data" (lange Binärdaten) ausgegeben.

Bitfolgen in SQL-92 (1)

- `BIT`(n): Bitfolgen mit genau n Bits.

  Konstanten von `BIT`(n) werden entweder binär, z.B. `B'11000101'`, oder hexadezimal, z.B. `X'C5'`, geschrieben. Es gibt auch `BIT VARYING`(n).

- Bitfolgen waren in SQL-86 nicht enthalten und werden von keinem der fünf Systeme unterstützt.

  Jedes System erlaubt jedoch binäre Daten und SQL Server und Access haben sogar einen Typ `BIT` (ohne Länge).

Bitfolgen in SQL-92 (2)

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

  SQL Server hat `BIT`, Access hat `YESNO`. Oracle und DB2 haben keine spezielle Unterstützung für boolesche Werte. Meist wird `CHAR(1)` zusammen mit der Bedingung, dass in dieser Spalte nur `'J'` und `'N'` erlaubt sind, verwendet. MySQL behandelt `BIT` und `BOOL` als Synonyme für `CHAR(1)` (ohne Bedingung). In Access und MySQL können die booleschen Spalten als Bedingungen verwendet werden.

Binäre Daten (1)

Oracle:

- `RAW`(n): Binäre Daten mit der Länge von n Bytes.

  n muss zwischen 1 und 2000 liegen.

- Binäre Daten werden z.B. für Grafiken verwendet.

  Immer wenn die Bedeutung der Daten von einem externen Programm interpretiert wird und die Datenbank die Bedeutung nicht kennt.

- Gewöhnlich konvertiert Oracle die Daten, wenn der Nutzer (Client, z.B. PC) einen anderen Zeichensatz als der Server hat. Das wird bei `RAW`-Daten nicht gemacht.

Binäre Daten (2)

Oracle, fortgesetzt:

- `RAW`-Daten werden in SQL-Statements als Zeichenketten mit hexadezimalen Ziffern geschrieben.
- `LONG RAW`: Binäre Daten variabler Länge, bis zu 2GB.
- `BLOB`: Binary large object (großes binäres Objekt), bis zu 4GB.

Binäre Daten (3)

DB2:

- Man kann für String-Spalten festlegen, dass sie binäre Daten enthalten, z.B. (Spalte `ENCRKEY`):

```
ENCRKEY VARCHAR(100) FOR BIT DATA
```

- Dies stellt sicher, dass
  - ◇ Vergleiche die exakten binären Codes benutzen,

    Es wird also die "collation sequence" nicht benutzt, die Kleinbuchstaben mit den entsprechenden Großbuchstaben identifizieren könnte.
  - ◇ keine Konvertierung zwischen verschiedenen Zeichensätzen ("code pages") erfolgt.

Binäre Daten (4)

DB2, fortgesetzt:

- String-Konstanten können in hexadezimaler Notation geschrieben werden, z.B. `X'FFFF'`.
- Es gibt auch große binäre Objekte, `BLOB`(n), die binäre Daten bis zu 2GB enthalten können.

Binäre Daten (5)

SQL Server:

- `BINARY`(n): Binäre Daten fester Länge von n Bytes.

  Die Größe n darf maximal 8000 sein.
  Eingabedaten werden mit `0x00` Bytes bis zur Länge n aufgefüllt.
- `VARBINARY`(n): Binäre Daten variabler Länge.

  n ist die maximale Länge in Bytes. Es kann maximal 8000 sein.
- Konstanten werden in der Form `0xFF1C` geschrieben.
- `IMAGE` kann binäre Daten bis zu 2GB speichern.

  Das ist der BLOB-Typ von SQL Server. Trotz seines Namens interpretiert SQL Server die enthaltenen Daten nicht, es speichert z.B. kein Grafik-Format für das Bild (GIF, JPEG, PNG, usw.).

Binäre Daten (6)

MySQL:

- `CHAR(`n`) BINARY`: String fester Länge von n Bytes.

  Wird `BINARY` an den String-Datentyp angehängt, werden Vergleiche durchgeführt, indem die Byte-Codes auf exakte Gleichheit verglichen werden. Ohne `BINARY` werden Vergleiche case-insensitiv durchgeführt (Groß- und Kleinbuchstaben werden identifiziert).

- `VARCHAR(`n`) BINARY`: String variabler Länge $\leq n$ Bytes.
- `BLOB`, `MEDIUMBLOB`, `LONGBLOB`: siehe oben.
- Konstanten kann man hexadezimal schreiben.

  Z.B. `0x612D7A`. Ansonsten müssen beim Einfügen binärer Daten die Bytes 0 (ASCII NUL), 34 (ASCII "), 39 (ASCII '), 92 (ASCII \) mit Escape-Sequenzen codiert werden: (\0, \", \', \\).

Binäre Daten (7)

Access:

- `BINARY`, `BINARY(`n`)`: Byte-String.

  In Access 2000 muss $n \leq 510$ gelten.

- Vergleiche mit diesem Typ sind case-sensitiv.
- `OLEOBJECT`, `LONGBINARY`: siehe oben.
- Hexadezimale Konstanten der Form `0x61002D007A00` sind möglich.

  Da Access Unicode verwendet, muss man zwei Bytes pro Zeichen schreiben. Das obige Beispiel ist der String 'a-z'. Man beachte, dass die Bytes eines 16-Bit-Integers "vertauscht" sind. Die Bedingung `'a'=0x6100` wird als wahr ausgewertet.

Datums- und Zeit-Typen (1)

SQL-92:

- `DATE`: Ein Wert zwischen 0001-01-01 (1. Jan. 0001) und 9999-12-31 (31. Dez. 9999).

  Natürlich sind ungültige Daten wie 1999-02-29 ausgeschlossen. `DATE`-Konstanten werden als Zeichenkette der Form `YYYY-MM-DD` geschrieben, gekennzeichnet mit dem Schlüsselwort `DATE`, z.B. `DATE '1965-06-26'`.

- `TIME`: Zeit (von 00:00:00 bis 23:59:59).

  Man kann auch Bruchteile einer Sekunde speichern. Z.B. erlaubt `TIME(3)` das Speichern von Werten wie `16:20:31.001`. Der Sekunden-Anteil kann bis 61.9 gehen (für Schaltsekunden). `TIME`-Konstanten werden als Zeichenketten der Form HH:MM:SS[.SSS] geschrieben, wobei das Wort "`TIME`" vorangestellt wird, z.B. `TIME '09:30:00'`.

- SQL-92 unterstützt auch verschiedene Zeitzonen.

Datums- und Zeit-Typen (2)

SQL-92, fortgesetzt:

- `TIMESTAMP`: `DATE` und `TIME(6)` zusammen.

  Z.B.: `TIMESTAMP '1999-03-23 18:30:00.000000'`.

- `INTERVAL DAY(`p`)`: Zeitintervall in Tagen.

  Werte sind n Tage, $-10^p < n < 10^p$. `INTERVAL DAY(3)` ist eine Differenz zwischen zwei `DATE`-Werten (positiv oder negativ), die 999 Tage nicht überschreiten kann. Eine Konstante ist z.B. `INTERVAL '14' DAY`.

- `INTERVAL HOUR(`p`) TO SECOND`: Differenz zwischen `TIME`-Werten in Stunden ($< 10^p$), Minuten, Sekunden.

  Eine Konstante ist z.B. "`INTERVAL '2:12:35' HOUR TO SECOND`". Anstelle von "`HOUR TO SECOND`" kann man z.B. "`DAY TO MINUTE`" oder eine beliebige andere Genauigkeit angeben.

Datums- und Zeit-Typen (3)

DB2:

- DB2 unterstützt `DATE`, `TIME` und `TIMESTAMP`.

  `TIME` ist immer in Sekunden, eine Präzision kann man nicht festlegen. `TIMESTAMP` hat jedoch Mikrosekunden. Es gibt keine spezifischen Konstanten für Datums- und Zeit-Werte (z.B. wird `TIME '09:30:00'` nicht verstanden), aber Zeichenketten bestimmter Formate werden automatisch konvertiert.
  `DATE: '2000-03-27', '03/27/2000', '27.03.2000'.`
  `TIME: '09:30:00', '9:30', '09.30.00', '9:30 AM'.`
  `TIMESTAMP: '2000-03-27-09.30.00.000000'.`

- DB2 hat keinen `INTERVAL`-Typ.

  Aber gewisse Intervalle können als Argumente von + und – verwendet werden. Z.B. funktioniert `DUE_DATE + 21 DAYS < CURRENT DATE`, aber `CURRENT DATE - DUE_DATE > 21 DAYS` ist ungültig.

Datums- und Zeit-Typen (4)

Oracle:

- Oracle unterstützt die SQL-92-Typen nicht.
- Oracle hat einen Typ für Zeitstempel: `DATE`.

  Trotz seines Namens speichert `DATE` auch die Zeit (in Stunden, Minuten, Sekunden). Wird nur ein Datum festgelegt, geht Oracle von der Uhrzeit 00:00:00am (Mitternacht, Tagesbeginn) aus.

- Es gibt keine spezifischen `DATE`-Konstanten, aber Oracle konvertiert Strings automatisch.

  Strings der Form `'DD-MON-YY'` (z.B. `'23-JAN-99'`) werden akzeptiert, wenn Datumswerte benötigt werden. Man verwende `TO_DATE` u. `TO_CHAR` für andere Formate (einschließlich Zeit). Das Default-Format hängt von `NLS_DATE_FORMAT` ab: In Deutschland wird `'DD.MM.YY'` verwendet.

Datums- und Zeit-Typen (5)

SQL Server:

- `DATETIME`: Vom 1. Jan 1753 bis zum 31. Dez 9999.

  Datum und Zeit (wie Oracles `DATE`). Genauigkeit: 0.003s.

- `SMALLDATETIME`: Vom 1. Jan 1900 bis 6. Juni 2079.

  Genauigkeit: 1 Minute. Benötigt 4 Bytes (`DATETIME`: 8 Bytes).

- Es gibt keine spezifischen `DATETIME`-Konstanten, aber Strings werden automatisch transformiert.

  Das Default-Ausgabeformat ist `'2000-03-29 18:00:00'`. SQL Server versteht jedoch auch andere Formate, z.B. `'March 29, 2000'` (fehlt die Zeit, wird 00:00 angenommen), `'29-MAR-2000 12:00'`, `'03/27/00 9:00 PM'`, `'14:30:00'` (fehlt das Datum, wird 01.01.1900 angenommen).

Datums- und Zeit-Typen (6)

MySQL:

- `DATETIME`: Datum und Zeit (sekundengenau).

  Von `'1000-01-01 00:00:00'` bis `'9999-12-31 23:59:59'`. Das normale Format für Konstanten ist `'YYYY-MM-DD HH:MM:SS'`. Einige alternative Formate werden unterstützt (aber Jahr immer zuerst). MySQL lässt ungültige Daten wie `'2002-02-31 00:00:00'` zu und Jahr, Monat und Tag können Null sein (was eine Art partiellen Nullwert ergibt).

- `DATE`: Datum (von `'1000-01-01'` bis `'9999-12-31'`).
- `TIME`: Uhrzeit und Zeitintervall.

  Von `-838:59:59` bis `838:59:59` (mehr als 34 Tage rückwärts bis 34 Tage vorwärts). Man muss Sekunden in `TIME`-Konstanten einfügen, z.B.wird `'06:15'` als `'00:06:15'` verstanden. Formate sind z.B. `'HH:MM:SS'`, `'HHH:MM:SS'` oder `'DD HH:MM:SS'` (`DD` sind Tage zwischen 0 und 33).

Datums- und Zeit-Typen (7)

MySQL, fortgesetzt:

- `YEAR`

  Von 1901 bis 2155 (1 Byte). Das normale Format ist 'YYYY'. Zweistellige Jahreszahlen von 70 bis 99 werden in 1970 bis 1999 konvertiert und 00 bis 69 werden als 2000 bis 2069 verstanden.

- `TIMESTAMP`: Datum und Zeit (in Sekunden).

  Werte zwischen 1970 und irgendwann im Jahr 2037. MySQL behandelt eine Spalte dieses Typs auf besondere Art: Sie hat als Default-Wert automatisch das aktuelle Datum/Zeit (bei mehreren `TIMESTAMP`-Spalten gilt dies nur für die erste). Somit wird in der `TIMESTAMP`-Spalte das Datum und die Zeit der Erstellung einer neuen Zeile gespeichert, wenn kein anderer Wert für diese Spalte festgelegt wurde. `TIMESTAMP`-Werte werden als Integer dargestellt, z.B. im Format `YYYYMMDDHHMMSS`. Man kann eine Ausgabe-Größe festlegen, z.B. `TIMESTAMP(8)`: `YYYYMMSS`.

Datums- und Zeit-Typen (8)

Access:

- `DATETIME`: Datum und Zeit zwischen den Jahren 100 und 9999 (sekundengenau).

  Als Gleitkommazahl gespeichert: Der ganzzahlige Teil ist die Anzahl der Tage seit dem 30. Dez. 1899. Der gebrochene Teil ist die Anzahl der Sekunden seit Mitternacht.
  Wahrscheinlich sind Jahre vor 100 ausgeschlossen, um zweistellige Jahresangaben erkennen zu können.

- Konstanten werden z.B. in der Form `#MM-DD-YYYY#` oder `#MM-DD-YYYY HH:MM:SS#` geschrieben.

  Man kann auch "/" anstelle von "-" verwenden oder das Jahr zweistellig angeben.

Nationale Sprachen (1)

- Oracle kann spezielle deutsche Buchstaben wie `ä`, `ö`, `ü` oder `ß` speichern und es kann z.B. auch mit japanischen Zeichen arbeiten.

  Oracle transformiert Zeichen zwischen verschiedenen Kodierungsschemata, z.B. in einer Client/Server-Umgebung.

- Oracle kann Fehlermeldungen etc. in verschiedenen Sprachen ausgeben.

  Auch Dinge wie eine richtige Sortierungs-Reihenfolge und das Format für Datumswerte etc. kann an nationale Standards angepasst werden.

- Manche Entscheidungen, wie der DB-Zeichensatz, müssen während der Installation getroffen werden.

Nationale Sprachen (2)

SQL-92:

- Der SQL-92-Standard enthält auch einen großen Abschnitt über nationale Zeichensätze (der sich von Oracle unterscheidet).

SQL Server:

- Während der Installation von SQL Server kann ein Zeichensatz für `CHAR`, `VARCHAR`, `TEXT` gewählt werden.
- `NCHAR`, `NVARCHAR`, `NTEXT` speichern Strings in Unicode (2 Bytes je Zeichen). Konstanten z.B. N'aäb'

Nationale Sprachen (3)

DB2:

- Das CREATE DATABASE-Statement hat die Optionen CODESET und TERRITORY.
- Auf diese Weise wird ein Zeichensatz (ggf. auch mit 2 Bytes pro Zeichen) festgelegt.
- Die Typen GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC und DBCLOB Zeichen mit 2 Byte Codes.

Access:

- Access verwendet Unicode für Text-Typen.

Andere Datentypen (1)

Oracle:

- BFILE: Referenz zu einer Betriebssystem-Datei.

  Die Datei selbst wird nicht in der DB gespeichert, nur ihr Name. Im Gegensatz zu CLOB und BLOB findet keine Transaktionsverwaltung statt. Externe Dateien können über die Datenbank nur gelesen werden.

- ROWID: Physischer Zeiger auf eine bestimmte Zeile.

  Die ROWID spezifiziert Datei, Block und Tupelnummer. Zugriffe über die ROWID sind sehr schnell. Jede Tabelle hat eine "Pseudo-Spalte" ROWID (sie kann wie eine normale Spalte unter SELECT und WHERE verwendet werden). ROWID-Komponenten werden z.B. mit DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) angezeigt.

- Nutzer-definierte PL/SQL Datentypen.

Andere Datentypen (2)

SQL Server:

- MONEY und SMALLMONEY: Zahlen mit einer Genauigkeit von 1/10000 einer Geldeinheit.

  SMALLMONEY sind 32-Bit-Zahlen von -214748.3648 bis +214748.3647, MONEY sind 64-Bit (bis zu 922 Mrd.). Eine Konstante ist z.B. $12.34.

- TIMESTAMP: DB-weit eindeutige Zahl, automatisch erzeugt und bei jedem Update der Zeile geändert.
- UNIQUEIDENTIFIER: Global eindeutiger Bezeichner.
- CURSOR: Referenz zu einem Cursor.

  Das ist eine SQL-Anfrage (die möglicherweise gerade ausgeführt wird) oder ein Anfrage-Resultat.

Andere Datentypen (3)

DB2:

- DATALINK: Referenz zu einer Datei, die außerhalb der DB gespeichert ist (URL).

Access:

- CURRENCY: Zahlen mit 4 Stellen nach dem Komma.

  −922 337 203 685 477.5808 bis 922 337 203 685 477.5807 (64 Bit-Zahl). Mit Währungssymbol angezeigt (z.B. $10.25).

- GUID: Vom System generierte eindeutige Zahl.

  16 Bytes. Man sollte nicht in Spalten dieses Typs schreiben.

Andere Datentypen (4)

MySQL:

- `ENUM`($v_1, v_2, \ldots$): Einer der Werte v_i.

  Die Werte werden als String-Konstanten, z.B. `ENUM('MO','DI',...)`, geschrieben. Ein Aufzählungstyp kann bis zu 65 535 verschiedene Werte haben. Wird ein ungültiger Wert (nicht in $\{v_1, v_2, \ldots\}$) eingefügt, fügt MySQL stattdessen den leeren String ein. Die Werte sind durchnummeriert, der leere String hat die Zahl 0. MySQL erlaubt Vergleiche mit Zahlen, man kann auch Operationen für Zahlen anwenden (z.B. +). MySQL sortiert Aufzählungs-Werte nach ihrem numerischen Wert, d.h. in der Reihenfolge, in der sie deklariert wurden.

- `SET`($v_1, v_2, \ldots$): Jede Teilmenge von $\{v_1, v_2, \ldots\}$.

  Werte sind wieder String-Konstanten. Eine Menge kann max. 64 Elemente haben. Mengenkonstanten werden als String mit den Element-Namen (durch Kommas getrennt) geschrieben (in v_i keine Kommas).

Inhalt

1. Klassische SQL-Datentypen
2. Weitere SQL-Datentypen
3. CREATE TABLE-Syntax
4. CREATE SCHEMA, DROP TABLE
5. ALTER TABLE

Beispiel (1)

STUDENTEN

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

BEWERTUNGEN

<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	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

<u>ATYP</u>	<u>ANR</u>	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

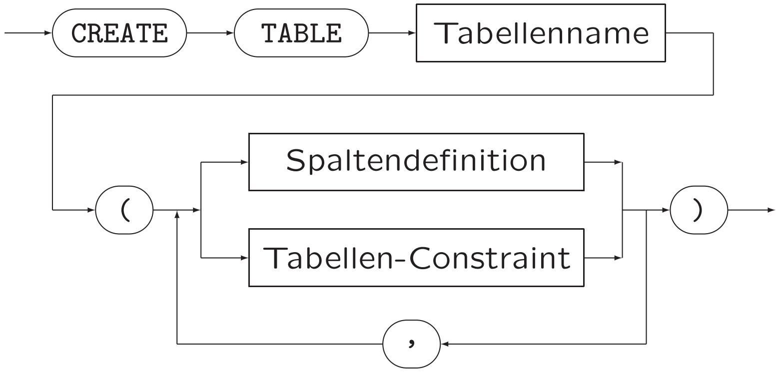
Beispiel (2)

- Das `CREATE TABLE`-Statement in SQL definiert:
  - ◇ Tabellenname
  - ◇ Spalten und ihre Datentypen
  - ◇ Constraints (`NOT NULL`, (Fremd-)Schlüssel, `CHECK`)
- Z.B. `STUDENTEN(`<u>`SID`</u>`, VORNAME, NACHNAME, EMAIL`^{o}`)`:

```
CREATE TABLE STUDENTEN (
       SID      NUMERIC(3)  NOT NULL PRIMARY KEY
                            CHECK(SID > 0),
       VORNAME  VARCHAR(20) NOT NULL,
       NACHNAME VARCHAR(20) NOT NULL,
       EMAIL    VARCHAR(128),
       UNIQUE(VORNAME, NACHNAME) )
```

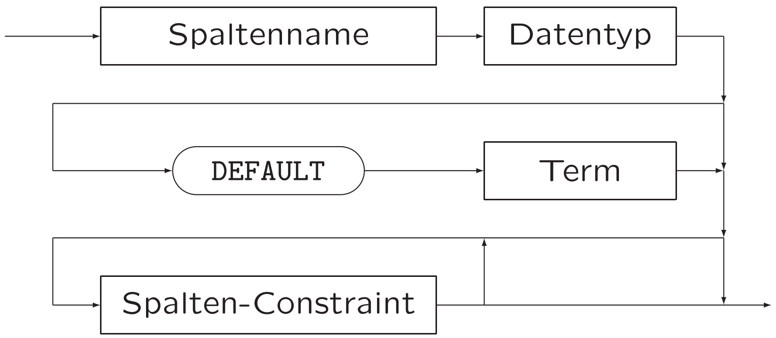
CREATE TABLE: Syntax (1)

CREATE TABLE-Statement:



CREATE TABLE: Syntax (2)

Spaltendefinition:



Constraints: Überblick (1)

- In SQL kann man Constraints als Spalten-Constraints oder Tabellen-Constraints definieren.
- Spalten-Constraints sind Bedingungen, die sich nur auf eine Spalte beziehen. Tabellen-Constraints können sich auf mehrere Spalten beziehen.
- Spalten-Constraints (außer vielleicht `NOT NULL`) können auch als "Tabellen-Constraints" formuliert werden, aber die Syntax von Spalten-Constraints ist etwas einfacher.

  Intern werden Spalten-Constraints in Tabellen-Constraints übersetzt.

Constraints: Überblick (2)

- Spalten-Constraints werden in der Spaltendefinition festgelegt (nur durch Leerzeichen getrennt).
- Spalten-Constraints im Beispiel:

```
CREATE TABLE STUDENTEN (
       SID      NUMERIC(3)   NOT NULL  PRIMARY KEY
                             CHECK(SID>0),
       VORNAME  VARCHAR(20)  NOT NULL,
       NACHNAME VARCHAR(20)  NOT NULL,
       EMAIL    VARCHAR(128),
       UNIQUE(VORNAME, NACHNAME) )
```

Constraints: Überblick (3)

- Tabellen-Constraints werden durch ein Komma von den Spaltendefinitionen und voneinander getrennt:
- Tabellen-Constraint im Beispiel:

```
CREATE TABLE STUDENTEN (
       SID      NUMERIC(3)  NOT NULL PRIMARY KEY
                            CHECK(SID>0),
       VORNAME  VARCHAR(20) NOT NULL,
       NACHNAME VARCHAR(20) NOT NULL,
       EMAIL    VARCHAR(128),
       UNIQUE(VORNAME, NACHNAME) )
```


Constraints: Überblick (4)

- Gleiches Beispiel, aber Spalten-Constraints (außer NOT NULL) durch Tabellen-Constraints ersetzt:

```
CREATE TABLE STUDENTEN (
       SID      NUMERIC(3)  NOT NULL,
       VORNAME  VARCHAR(20) NOT NULL,
       NACHNAME VARCHAR(20) NOT NULL,
       EMAIL    VARCHAR(128),
       PRIMARY KEY (SID),
       CHECK(SID > 0),
       UNIQUE(VORNAME, NACHNAME) )
```


Spalten-Constraints (1)

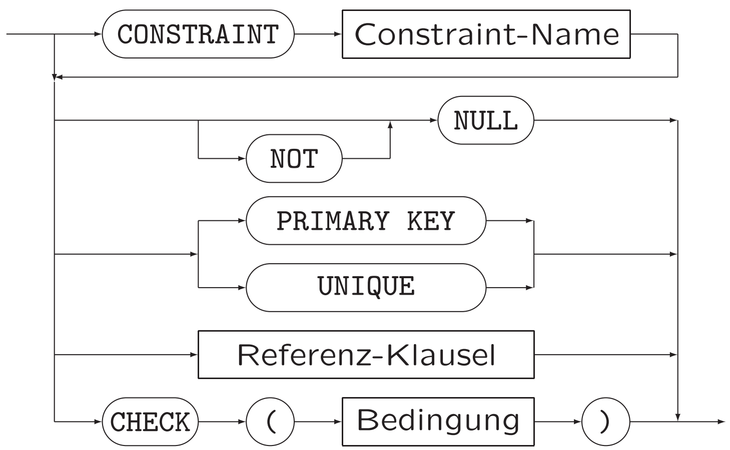
- Es gibt fünf Arten von Spalten-Constraints:
  - ◇ `NOT NULL`: Keine Nullwerte in dieser Spalte.
  - ◇ `PRIMARY KEY`: Diese Spalte ist der Primärschlüssel der Tabelle.
  - ◇ `UNIQUE`: Diese Spalte ist ein Alternativschlüssel.
  - ◇ `REFERENCES` T: Die Spalte ist ein Fremdschlüssel.

    D.h. Werte dieser Spalte müssen in der Primärschlüssel-Spalte der Tabelle T auftauchen.
  - ◇ `CHECK` (C): Werte der Spalte müssen C erfüllen.

    C ist eine Bedingung wie in der `WHERE`-Klausel, nur mit gewissen Einschränkungen (siehe unten).

Spalten-Constraints (2)

Spalten-Constraint:



Spalten-Constraints (3)

- Man kann "`NULL`" als Spalten-Constraint schreiben, um zu betonen, dass Nullwerte erlaubt sind.

  Das ist jedoch keine richtige Bedingung und ist sowieso der Default.

- `PRIMARY KEY` impliziert `NOT NULL`.

  DB2 verlangt jedoch, dass `NOT NULL` zusätzlich festgelegt wird.
  In MySQL muss `NOT NULL` explizit deklariert werden, wenn der Primärschlüssel als Tabellen-Constraint definiert wurde.

- Es darf nur einen Primärschlüssel je Tabelle geben.
- `UNIQUE` impliziert `NOT NULL` nicht.

  In DB2 kann `UNIQUE` nur zusammen mit `NOT NULL` verwendet werden.

Spalten-Constraints (4)

- SQL-86 unterstützte nur [`NOT NULL` [`UNIQUE`]].
- Außerdem war `UNIQUE` in vielen Systemen nicht implementiert.
- In älterem Code wurde oft "`CREATE UNIQUE INDEX`" verwendet, um Schlüsselbedingungen zu erzwingen.

  Siehe Teil 15 über physischen Entwurf.

- 1989 wurde der Standard um ein optionales "Integrity Enhancement Feature" erweitert (SQL-89).

  Dies enthielt die obigen Konstrukte.

CHECK-Constraints (1)

- Die Bedingung C eines `CHECK`-Constraints sieht wie eine `WHERE`-Bedingung ohne Unteranfragen aus. Dabei sind Funktionen wie z.B. `SYSDATE`, die sich später ändern können, ausgeschlossen.

  In Spalten-Constraints kann nur diese eine Spalte verwendet werden.
  Ansonsten verwendet man Tabellen-Constraints (siehe unten).

CHECK-Constraints (2)

- Grundidee effizienter Constraint-Überprüfung: Bedingung war vor dem Update erfüllt.
- Das DBMS prüft nur geänderte/eingefügte Zeilen.

  Da solche Constraints im leeren DB-Zustand gelten und das System sicherstellt, dass Updates deren Gültigkeit nicht zerstören, folgt per Induktion, dass sie in jedem DB-Zustand gelten. Das erklärt auch, warum `SYSDATE` verboten ist: Constraints könnten ohne Update ungültig werden.

- `CHECK`-Constraints werden von MySQL und Access nicht unterstützt.

CHECK-Constraints (3)

- Oracle, SQL Server und DB2 schließen Unteranfragen in `CHECK`-Constraints aus.

  Somit ist eine grundlegende Einschränkung in `CHECK`-Constraints, dass sie für jedes einzelne Tupel getrennt auswertbar sein müssen.

- SQL-92 erlaubt Unteranfragen innerhalb von `CHECK`-Constraints, aber dann ist die Integritätsprüfung schwierig/ineffizient.

  Wird eine in der Unteranfrage verwendete Tabelle geändert, ist es im allgemeinen schwierig herauszufinden, für welche der Zeilen die `CHECK`-Bedingung erneut überprüft werden muss. Eine einfache Lösung wäre, alle Zeilen zu überprüfen, aber dann führt jede kleine Änderung zu einer langen Integritätsprüfung.

CHECK-Constraints (4)

- Wären Unteranfragen unter `CHECK` implementiert, könnte ein Fremdschlüssel so formuliert werden:

```
CREATE TABLE BEWERTUNGEN( Nicht implementiert!
  SID NUMERIC(3)
      CHECK(SID IN (SELECT SID FROM STUDENTEN)),
  ... )
```

- Wenn ein `STUDENTEN`-Tupel t gelöscht oder seine `SID` geändert wird, betrifft dies nur Tupel in `BEWERTUNGEN`, die die (alte) SID von t haben.

  Es wäre dann also nicht nötig, die `CHECK`-Bedingung in `BEWERTUNGEN` für alle Tupel zu überprüfen.

Constraints und Nullwerte

- Integritätsbedingungen gelten als erfüllt, wenn das Ergebnis den Wahrheitswert "unbekannt" hat.

  Die Definitionen für Schlüssel und Fremdschlüssel, die aus mehreren Spalten bestehen, von denen nur einige Null sind, sind kompliziert und systemabhängig.

- Z.B. kann bei dieser Deklaration die E-Mail-Adresse Null sein. Ist sie es nicht, muss sie "`@`" enthalten:

```
CREATE TABLE STUDENTEN(
   ...,
   EMAIL VARCHAR(128)
         CHECK(EMAIL LIKE '%@%'))
```


Constraint-Namen (1)

- Einem Constraint kann ein Name gegeben werden, indem man "`CONSTRAINT` ⟨`Name`⟩" voranstellt.

  MySQL erlaubt Constraint-Namen nur für Tabellen-Constraints. Es scheint sie jedoch sowieso gleich wieder zu vergessen.

- Constraint-Namen müssen innerhalb eines Schemas eindeutig sein, wohingegen Spaltennamen nur eindeutig für eine Tabelle sein müssen.

  In DB2 müssen Constraint-Namen nur für eine Tabelle eindeutig sein.
  In DB2 können `NOT NULL`-Constraints nicht benannt werden.

- Definieren Sie immer Namen (außer für `NOT NULL`). Sonst wählt das System Namen wie "`SYS_C036`".

Constraint-Namen (2)

- Wenn ein Constraint verletzt ist, wird der Name ausgegeben: System-generierte Namen ergeben unverständliche Fehlermeldungen.

  Die Fehlermeldung für Not Null-Constraints ist meist klar:
  `ORA-01400: cannot insert NULL into (USER.TABLE.COLUMN)`
  Gibt es mehrere Schlüssel, ist dies schon unklar:
  `ORA-00001: unique constraint (BRASS.SYS_C007916) violated`
  Für einen Fremdschlüssel bekommt man diese Fehlermeldung:
  ```
  ORA-02291: integrity constraint (BRASS.SYS_C007914) violated -
             parent key not found.
  ```
  Für Check-Constraints erhält man diese Nachricht:
  `ORA-02290: check constraint (BRASS.SYS_C007915) violated.`

- Namen erleichtern das Löschen von Constraints.

Constraint-Namen (3)

- Beispiel mit Constraint-Namen:

```
CREATE TABLE STUDENTEN (
       SID      NUMERIC(3) NOT NULL
                CONSTRAINT SID_MUSS_POSITIV_SEIN
                           CHECK(SID > 0)
                CONSTRAINT STUDENTEN_SCHLUESSEL
                           PRIMARY KEY,
       VORNAME  VARCHAR(20) NOT NULL,
       NACHNAME VARCHAR(20) NOT NULL,
       EMAIL    VARCHAR(128),
       CONSTRAINT STUDENTENNAMEN_EINDEUTIG
                  UNIQUE(VORNAME, NACHNAME) )
```


Tabellen-Constraints (1)

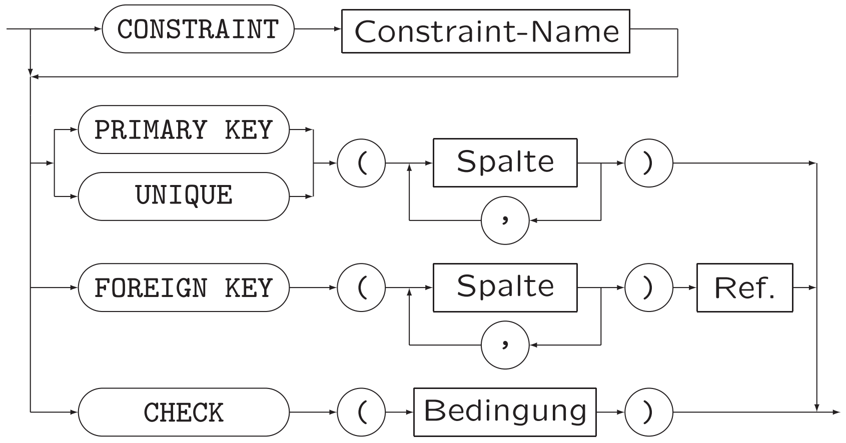
- Tabellen-Constraints werden benötigt, wenn ein (Fremd-)Schlüssel mehrere Spalten hat oder sich eine CHECK-Bedingung auf mehrere Spalten bezieht.

  Constraints, die sich nur auf eine Spalte beziehen, können als Spalten- oder Tabellen-Constraint definiert werden.

- Die vier Arten von Tabellen-Constraints sind:
  - ◇ PRIMARY KEY(A_1, ..., A_2)
  - ◇ UNIQUE(A_1, ..., A_2)
  - ◇ FOREIGN KEY(A_1, ..., A_2) REFERENCES R
  - ◇ CHECK(C)

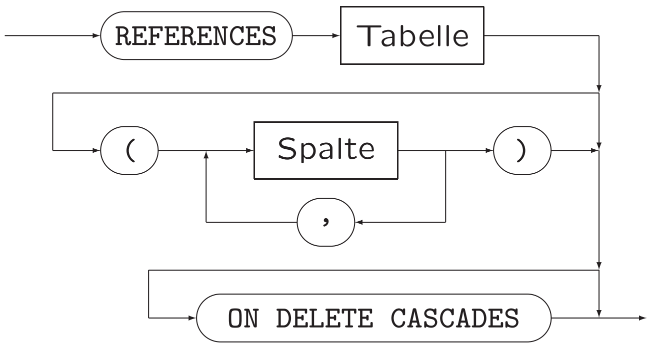
Tabellen-Constraints (2)

Tabellen-Constraint:



Fremdschlüssel (1)

Referenz-Klausel (Ref.):



Fremdschlüssel (2)

- Die Referenz-Klausel steht in Spalten-Constraints hinter der Spalte und in Tabellen-Constraints hinter der `FOREIGN KEY`-Deklaration.
- Es ist möglich, die referenzierten Spaltennamen anzugeben, z.B. `REFERENCES STUDENTEN(SID)`.

  Es können jedoch nur Schlüssel (`PRIMARY KEY` oder `UNIQUE`) referenziert werden. Werden keine Spalten genannt, wird der Primärschlüssel referenziert. Es macht selten Sinn, Alternativschlüssel zu referenzieren.
- Fremdschlüssel und referenzierter Schlüssel müssen die gleiche Spaltenanzahl und zusammengehörige Spalten den gleichen Datentyp haben.

Fremdschlüssel (3)

- MySQL prüft keine Fremdschlüssel, erlaubt aber die Deklaration von Fremdschlüsseln in `CREATE TABLE`.
- Beispiel: BEWERTUNGEN(<u>SID</u>→STUDENTEN, ...)
  - ◇ In SQL-92, Oracle, DB2 (nicht in SQL Server, Access) kann man verlangen, dass die Bewertungen eines Studenten automatisch mitgelöscht werden, wenn man ihn löscht:

    `REFERENCES STUDENTEN ON DELETE CASCADES`
  - ◇ Sonst lehnt das DBMS die Löschung eines Studenten ab, von dem es noch Bewertungen gibt.

Fremdschlüssel (4)

- Gegeben sei eine DB mit Vorlesungsdaten:
  VORLESUNGEN(..., DOZENT^{o}→PROFESSOREN, ...).
- In SQL-92 und DB2 (nicht in Oracle, SQL Server, Access) kann man festlegen, dass `DOZENT` auf Null gesetzt wird, wenn der Professor gelöscht wird. Die Vorlesungen bleiben dann in der DB.
- Dazu schreibt man: “`ON DELETE SET NULL`”.
- In SQL-92 kann man auch “`SET DEFAULT`” wählen. Das wird in keinem der fünf Systeme unterstützt.

Fremdschlüssel (5)

- In SQL-92 kann man auch die Reaktion auf Änderungen des Schlüssels von `PROFESSOREN` festlegen (Namensänderung).
- Dazu schreibt man "`ON UPDATE ...`".
  Es wird von keinem der fünf Systeme unterstützt.

  DB2 versteht "`ON UPDATE`" mit den Parametern "`NO ACTION`" und "`RESTRICT`", aber das Ablehnen der Updates von referenzierten Schlüsselwerten ist der Default.

Default-Spaltenwerte (1)

- Im Befehl zum Erstellen neuer Zeilen (`INSERT`) muss man nicht für alle Spalten Werte definieren.

  `INSERT` wird in Kapitel 12 erläutert. Wird eine Zeile über eine Sicht eingefügt, kann es sein, dass man gar nicht für alle Spalten Werte angeben kann.
- Normalerweise wird in fehlenden Spalten ein Nullwert eingesetzt.
- Es ist jedoch möglich, einen Default-Wert festzulegen, der in Spalten, für die kein Wert gegeben ist, gespeichert wird.

Default-Spaltenwerte (2)

- Z.B. sollen alle Aufgaben `MAXPT = 10` haben, wenn der `INSERT`-Befehl keinen Wert für `MAXPT` enthält:

```
CREATE TABLE AUFGABEN(
       ...
       MAXPT NUMERIC(2) DEFAULT 10
                        NOT NULL
                        CHECK(MAXPT >= 0))
```

- "`DEFAULT SYSDATE`" speichert das aktuelle Datum in der Spalte (Datum der Zeilenerstellung).

  "`SYSDATE`" funktioniert nur in Oracle. In SQL Server und SQL-92: "`CURRENT_TIMESTAMP`". In DB2: "`CURRENT TIMESTAMP`". MySQL verwendet für die erste Spalte des Typs "`TIMESTAMP`" automatisch das aktuelle Datum als Defaultwert (man kann dies nicht explizit festlegen).

Default-Spaltenwerte (3)

- Manchmal können verschiedene Nutzer neue Zeilen in einer Tabelle einfügen. Mit "`DEFAULT USER`" wird der Name des aktuellen Nutzers in einer Spalte gespeichert (z.B. `EINGEGEBEN_VON`).
- Auf diese Weise kennt man den Nutzer, der für das Einfügen einer bestimmten Zeile verantwortlich ist.

  "`USER`" war schon im SQL-86-Standard enthalten und müsste sehr portabel sein. Aber MySQL und Access unterstützen "`USER`" nicht.

Default-Spaltenwerte (4)

- INSERT-Rechte können selektiv für bestimmte Spalten an Datenbank-Benutzer vergeben werden.
- Der Benutzer kann dann für die anderen Spalten bei der Tupel-Einfügung keine Werte angeben.
- Daher kann er/sie dann den DEFAULT-Wert nicht überschreiben.

  Natürlich muss für die UPDATE-Rechte die gleiche Einschränkung gelten. Der Benutzername oder das aktuelle Datum werden dann immer so gespeichert, wie es im Default-Wert definiert ist.

Default-Spaltenwerte (5)

- DEFAULT war in SQL-86 nicht enthalten und Access unterstützt es nicht.
- In SQL-92 darf der Default-Wert nur sein:
  - ◇ eine Konstante,
  - ◇ eine Funktion ohne Argumente oder

    Genauer: USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, CURRENT_DATE, CURRENT_TIME[(...)], CURRENT_TIMESTAMP[(...)]
  - ◇ NULL.
- Oracle akzeptiert jeden Term ohne Spaltennamen.
- MySQL erlaubt nur Konstanten als Default-Werte.

Default-Spaltenwerte (6)

- "DEFAULT NULL" wird verwendet, wenn kein Default-Wert explizit definiert ist.
- Somit ist es unmöglich, eine Zeile ohne festgelegten Wert für eine Spalte, die NOT NULL ist und keinen definierten Default-Wert hat, einzufügen.
- MySQL vermeidet dies, indem z.B. der leere String oder 0 als Default-Wert für NOT NULL-Spalten verwendet wird, wenn kein anderer definiert wurde.

  Dies verletzt den Standard und macht Fehlererkennung schwieriger. Man kann auch nicht explizit "DEFAULT NULL" für eine NOT NULL-Spalte definieren.

Eindeutige Zahlen (1)

- Oft muss man eindeutige Zahlen generieren, z.B. die SID für Studenten (künstlicher Primärschlüssel).
- SQL-92 hat keinen Mechanismus dafür.
- Theoretisch könnte man das aktuelle Maximum in der Tabelle abfragen, und eins dazu addieren.

  Alternativ könnte man auch eine Tabelle mit einer Zeile und Spalte erstellen, die das Maximum enthält. Das ist evtl. etwas schneller, obwohl man das Maximum auch mit einem B-Baum-Index schnell findet.
- Aber bei gleichzeitigen Nutzern wird dies schwierig, siehe Teil 12. Daher haben viele DBMS einen Mechanismus zum Generieren von eindeutigen Zahlen.

Eindeutige Zahlen (2)

Oracle:

- Oracle hat ein DB-Objekt “sequence” (Sequenz):

```
CREATE SEQUENCE STUD_IDS START WITH 101
```

- Man kann folgendermaßen eine Zahl abrufen und den in der Sequenz gespeicherten Wert erhöhen:

```
SELECT STUD_IDS.NEXTVAL FROM DUAL
```

  (DUAL ist eine Dummy-Tabelle mit einer Zeile.)

- Man kann STUD_IDS.NEXTVAL nicht als DEFAULT festlegen, aber man kann es im INSERT-Befehl verwenden.

Eindeutige Zahlen (3)

SQL Server:

- In SQL Server kann man das Wort IDENTITY zur Deklaration einer Integer-Spalte hinzufügen:

```
SID NUMERIC(3) IDENTITY NOT NULL PRIMARY KEY
```

- Werte dieser Spalte kann man nicht festlegen, der nächste Wert wird automatisch eingefügt.

  Sogar wenn man keine Spalten-Liste unter INSERT verwendet, wird die Spalte in der VALUES-Liste übersprungen. Besser: Spalten festlegen. IDENTITY kann nicht zusammen mit DEFAULT verwendet werden.

- Man kann Startwert und Schrittweite festlegen:

```
SID NUMERIC(3) IDENTITY(100,1) PRIMARY KEY
```


Eindeutige Zahlen (4)

DB2:

- In DB2 kann man festlegen, dass ein Spaltenwert z.B. von einem “Generator eindeutiger Zahlen” (unique number generator) berechnet wird:

```
SID NUMERIC(3)
    GENERATED ALWAYS AS IDENTITY(START WITH 101)
    NOT NULL PRIMARY KEY
```

- “GENERATED ALWAYS” heißt, dass man keinen Wert für diese Spalte im INSERT-Befehl festlegen kann.

  Die Alternative ist GENERATED BY DEFAULT. Beide Fälle schließen ein DEFAULT-Statement aus. Es gibt weitere Parameter für den Generator eindeutiger Zahlen, z.B. IDENTITY(START WITH 101, INCREMENT BY 1).

Eindeutige Zahlen (5)

MySQL:

- In MySQL kann man das Wort AUTO_INCREMENT zur Deklaration einer Integer-Spalte hinzufügen:

```
SID INT AUTO_INCREMENT PRIMARY KEY
```

- Dies funktioniert nur mit binären Integer-Typen.

  Z.B. funktioniert es nicht mit NUMERIC(3). Man kann UNSIGNED-Typen verwenden. Die Spalte muss als Schlüssel deklariert sein (PRIMARY KEY oder UNIQUE).

- Wird mit INSERT für diese Spalte NULL oder 0 festgelegt, wird stattdessen die nächste Zahl eingefügt.

  Ein deklarierter Default-Wert wird einfach ignoriert.

Eindeutige Zahlen (6)

Access:

- Access hat den Datentyp COUNTER, der eindeutige Zahlen erzeugt:

```
SID COUNTER NOT NULL PRIMARY KEY
```

- Man kann Startwert und Schrittweite mit Parametern dieses Datentyps festlegen:

```
SID COUNTER(100,1) NOT NULL PRIMARY KEY
```

- Um den nächsten Wert des Zählers einzufügen, muss man die Spalten unter `INSERT` explizit angeben und die Spalte `SID` weglassen.

Temporäre Tabellen

- SQL-92 und einige DBMS ermöglichen die Deklaration temporärer Tabellen, die
  - ◇ automatisch am Ende der Transaktion oder Sitzung gelöscht werden,

    Je nach System kann es sein, dass die Tabelle selbst nicht gelöscht wird, sondern nur all ihre Zeilen.
  - ◇ unsichtbar für andere (parallele) Sitzungen sind.

    Tabellen anderer Benutzer sind normalerweise unsichtbar (außer der Nutzer hat Zugriffsrechte erteilt). Bei temporären Tabellen haben aber auch parallele Sitzungen unter der gleichen Nutzerkennung verschiedene Kopien.
- Für Details siehe Handbuch des jeweiligen DBMS.

Speicherparameter

- In den meisten DBMS hat das "`CREATE TABLE`"-Statement viele Speicherparameter, die verändert werden können.
- Diese Parameter gehören zum physischen Schema, nicht zum konzeptuellen Schema.

  Es ist etwas unglücklich, dass hier beide Dinge vermischt werden.
- Der SQL-Standard enthält keine Definition, die zu dem physischen Schema gehört.
- Ist die Performance wichtig, sollte man die Bedeutung der Parameter verstehen und diese verändern.

Einschränkungen (1)

SQL Server:

- max. 1024 Spalten pro Tabelle
- max. 8060 Bytes pro Zeile

  Die Daten von `TEXT`-Spalten etc. werden separat gespeichert.
- Schlüssel können max. 16 Spalten enthalten.
- Schlüsselwerte können max. 900 Bytes haben.

  Das schränkt die Konkatenation von Spaltenwerten für alle Spalten ein.

Einschränkungen (2)

DB2:

- max. 500 Spalten pro Tabelle
- max. 4005 Bytes pro Zeile

  Einschließlich des Deskriptors von LOB-Spalten, aber nicht dessen Daten.
- Schlüssel können max. 16 Spalten enthalten.
- Schlüsselwerte können max. 255 Bytes haben.

Einschränkungen (3)

Oracle:

- max. 1000 Spalten pro Tabelle
- max. 32 Spalten pro Schlüssel
- Schlüsselwerte (oder Index-Werte) sind auf 40% der Blockgröße beschränkt.

  Die DB-Blockgröße kann innerhalb angemessener Grenzen konfiguriert werden (wenn die DB erstellt wird).
- Die Zeilenlänge ist nicht beschränkt, aber die Performance lässt nach, wenn Zeilen auf verschiedene Blöcke aufgeteilt werden müssen.

Einschränkungen (4)

Access:

- max. 255 Spalten pro Tabelle
- max. 10 Spalten pro Schlüssel
- max. 2000 Zeichen in einer Zeile

  Gilt nicht bei Memo- und OLE-Objekt-Spalten.
- Die maximale Größe einer Tabelle ist 1 GB.
- max. 2 GB pro Datenbank (.mdb-Datei)

  Man kann Verknüpfungen zu Tabellen in anderen Dateien einfügen, um die Grenzen zu überschreiten.

Inhalt

CREATE SCHEMA (1)

- In Oracle und SQL Server entsprechen sich DB-Accounts und Schemas 1:1.

  Benötigt ein Nutzer mehr als ein Schema, müssen mehrere Accounts für die gleiche Person erstellt werden.

- Tabellen werden im System durch ihren Namen und ihren Eigentümer identifiziert.

  In SQL Server: Server, Datenbank, Eigentümer, Name.

- In DB2 sind Schemas und Nutzer verschiedene Dinge, obwohl jedes Schema zu genau einem Nutzer gehört.

CREATE SCHEMA (2)

- Es gibt einen `CREATE SCHEMA`-Befehl, der jedoch nur benötigt wird, wenn sich zwei Tabellen gegenseitig referenzieren:

```
CREATE SCHEMA AUTHORIZATION ⟨Account⟩
   CREATE TABLE ABT(ABTNR NUMERIC(2) PRIMARY KEY,
      CHEF NUMERIC(4) REFERENCES ANG, ...)
   CREATE TABLE ANG(ANGNR NUMERIC(4) PRIMARY KEY,
      ABTNR NUMERIC(2) REFERENCES ABT, ...);
```

- Man beachte, dass die einzelnen `CREATE TABLE`-Statements nicht durch ";" getrennt werden.

CREATE SCHEMA (3)

- MySQL, Access unterstützen `CREATE SCHEMA` nicht.
- In Oracle, DB2, SQL Server kann `CREATE SCHEMA` die Befehle `CREATE TABLE`, `CREATE VIEW`, `GRANT` enthalten.

  In DB2 sind auch `COMMENT ON` und `CREATE INDEX` erlaubt.

- Schlägt ein Statement fehl, wird nichts ausgeführt.
- Da DB2 mehrere Schemas pro Nutzer zulässt, lautet die Syntax dort

  `CREATE SCHEMA` ⟨Name⟩ `AUTHORIZATION` ⟨Nutzer⟩

  Der Schema-Name und die Autorisierungs-Klausel sind beide optional, aber mindestens eines der beiden wird in DB2 benötigt.

Tabellen löschen (1)

- Tabellen werden mit folgendem Befehl gelöscht:

  `DROP TABLE ⟨Tabellenname⟩`

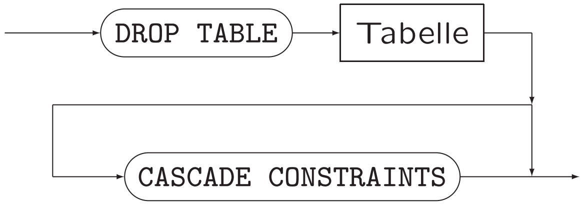
- Natürlich löscht dies auch alle Zeilen in der Tabelle.

  Man muss vorsichtig sein! In Oracle wird die aktuelle Transaktion automatisch beendet, wenn eine Tabelle gelöscht wird. Somit ist es nicht möglich, das Löschen oder vorangehende Aktionen rückgängig zu machen.

Tabellen löschen (2)

- Was passiert, wenn die Tabelle in Fremdschlüssel-Bedingungen referenziert wird?
  - ◇ In SQL Server und Access muss die referenzierende Tabelle zuerst gelöscht werden.
  - ◇ In DB2 wird der Fremdschlüssel automatisch gelöscht.
  - ◇ In Oracle kann "`CASCADE CONSTRAINTS`" angegeben werden, um Fremdschlüssel mit zu löschen.
  - ◇ In SQL-92 heißt es nur "`CASCADE`".

Tabellen löschen (3)



Beispiele:

- `DROP TABLE STUDENTEN CASCADE CONSTRAINTS`
- `DROP TABLE BEWERTUNGEN`

  Löscht man zuerst die Tabelle `BEWERTUNGEN` (enthält Fremdschlüssel) und dann `STUDENTEN`, so ist kein `CASCADE CONSTRAINTS` notwendig.

Inhalt

ALTER TABLE (1)

- Mit dem `ALTER TABLE`-Befehl kann man das Schema einer existierenden Tabelle verändern.
- Im Prinzip könnte man die Daten temporär woanders speichern, die Tabelle löschen, sie verändert neu erstellen und die Daten zurückkopieren. Aber
  - ◇ Kopieren ist unpraktisch bei großen Tabellen.
  - ◇ Tabelleneinträge könnten von Fremdschlüsseln referenziert werden.

    Dann muss man evtl. die gesamte DB neu erstellen. Auch Indexe, Grants, Sichten, Trigger etc. referenzieren Tabellen. Manches davon wird verlorengehen, wenn die Tabelle gelöscht wird.

ALTER TABLE (2)

- Beispiele für Änderungen eines Tabellen-Schemas:
  - ◇ Neue Spalten können hinzugefügt werden.

    Daher ist es sicherer, in Anwendungsprogrammen statt `SELECT *` Spaltennamen anzugeben.
  - ◇ Die Breite von Spalten kann erhöht werden.

    Z.B. von `VARCHAR(20)` zu `VARCHAR(30)`.
    Das ist eigentlich im SQL-92-Standard nicht möglich, aber in allen fünf DBMS (Oracle, DB2, SQL Server, Access, MySQL).
  - ◇ Constraints kann man hinzufügen/entfernen.

    In manchen Systemen kann man eine Bedingung (Constraint) auch deaktivieren, so dass sie nicht mehr überprüft wird, aber noch im System gespeichert ist. Später kann man sie wieder aktivieren.

ALTER TABLE (3)

- `ALTER TABLE` war nicht in SQL-86 enthalten.
- Es ist im SQL-92-Standard enthalten, aber DBMS-Implementierungen unterscheiden sich stark in der Syntax und darin, was geändert werden kann.
- Der SQL-92-Standard hat folgende Möglichkeiten:
  - ◇ Spalten können hinzugefügt oder entfernt werden.
  - ◇ Der Default-Wert einer Spalte kann geändert werden, aber der Datentyp nicht.
  - ◇ Constraints kann man hinzufügen/entfernen.

ALTER TABLE (4)

SQL-92 (Spalten hinzufügen):

- Z.B. Spalte "`ZUSATZPKT`" zu "`STUDENTEN`" hinzufügen:

```
ALTER TABLE STUDENTEN
      ADD COLUMN ZUSATZPKT NUMERIC(4,1)
                           CHECK(ZUSATZPKT >= 0)
```

- Das Schlüsselwort "`COLUMN`" ist optional.
- Die neue Spalte hat zunächst in allen Zeilen einen Nullwert.
- Der Wert kann anschließend verändert werden.

  Es kann jedoch zu Effizienzproblemen kommen, wenn die Zeilen viel länger werden, als sie beim Einfügen waren.

ALTER TABLE (5)

SQL-92 (Spalten hinzufügen, fortgesetzt):

- Wird ein Default-Wert festgelegt, kann der neue Wert `NOT NULL` sein:

```
ALTER TABLE STUDENTEN
  ADD ZUSATZPKT NUMERIC(4,1) DEFAULT 0 NOT NULL
```

- Die neue Spalte wird die letzte (rechts) in der Tabelle.

  Es ist nicht möglich, sie woanders einzufügen.
- Sichten (gespeicherte Anfragen) werden nicht beeinflusst, weil `SELECT *` durch eine explizite Spaltenliste ersetzt wird, wenn die Sicht gespeichert wird.

ALTER TABLE (6)

SQL-92 (Spalten entfernen):

- Spalten können aus Tabellen entfernt werden:

```
ALTER TABLE STUDENTEN
        DROP COLUMN ZUSATZPKT RESTRICT
```

- `RESTRICT` heißt, dass `ALTER TABLE` fehlschlägt, wenn die Spalte in Constraints/Sichten referenziert wird.

  Constraints, die sich nur auf diese Spalte beziehen, zählen nicht: Sie werden automatisch gelöscht.

- Die Alternative ist `CASCADE`: Das referenzierende DB-Objekt wird mit gelöscht.

  Es gibt keinen Default: Man muss `RESTRICT` oder `CASCADE` angeben.

ALTER TABLE (7)

SQL-92 (Spalten verändern):

- Die einzige erlaubte Veränderung einer Spalte ist die Änderung des Default-Wertes:

```
ALTER TABLE AUFGABEN
ALTER COLUMN MAXPT SET DEFAULT 12
```

- Der Default kann mit dieser Syntax auch auf Null gesetzt werden:

```
ALTER TABLE AUFGABEN
ALTER COLUMN MAXPT DROP DEFAULT
```

- In SQL-92 ist es nicht möglich, den Datentyp einer Spalte zu verändern.

ALTER TABLE (8)

SQL-92 (Constraints verändern):

- Eine Bedingung hinzufügen:

```
ALTER TABLE STUDENTEN
        ADD CONSTRAINT ZUSATZPKT_DEF
        CHECK(ZUSATZPKT IS NOT NULL)
```

  Es können nur Tabellen-Constraints hinzugefügt werden, aber Spalten-Constraints sind sowieso nur syntaktische Abkürzungen.

- Einen benannten Constraint entfernen:

```
ALTER TABLE STUDENTEN
      DROP CONSTRAINT ZUSATZPKT_DEF RESTRICT
```

  Date/Darwen behaupten, dass der Standard verlangt, `RESTRICT` oder `CASCADE` festzulegen, obwohl das nur bei Schlüsseln Sinn macht, die in Fremdschlüsseln referenziert werden.

ALTER TABLE (9)

Oracle (Spalten hinzufügen):

- Z.B. Spalte “`ZUSATZPKT`” zu “`STUDENTEN`” hinzufügen:

```
ALTER TABLE STUDENTEN ADD
  (ZUSATZPKT NUMERIC(4,1) CHECK(ZUSATZPKT >= 0))
```

- Bei existierenden Zeilen ist die neue Spalte Null.

  Natürlich kann man den Wert später verändern.

- Ist ein Default-Wert festgelegt, kann die neue Spalte `NOT NULL` sein:

```
ALTER TABLE STUDENTEN ADD
  (ZUSATZPKT NUMERIC(4,1) DEFAULT 0 NOT NULL
                          CHECK(ZUSATZPKT >= 0))
```

ALTER TABLE (10)

Oracle (Spalten verändern):

- Der Datentyp einer Spalte kann verändert werden:

```
ALTER TABLE AUFGABEN MODIFY (THEMA VARCHAR(100))
```

- Die Breite einer Spalte kann nicht verringert werden, außer sie enthält nur Nullwerte.
- MODIFY kann nicht verwendet werden, um Spalten-Constraints zu ändern, außer NULL/NOT NULL.

  Es gibt eine andere Syntax dafür, siehe nächste Folie.

- In Oracle können Spalten nicht gelöscht oder umbenannt werden.

ALTER TABLE (11)

Oracle (Constraints hinzufügen/entfernen):

- Einen Constraint hinzufügen:

```
ALTER TABLE STUDENTEN ADD
    (CONSTRAINT SCH2 UNIQUE(VORNAME, NACHNAME))
```

- Hier muss die Syntax für Tabellen-Constraints verwendet werden.

  Intern werden alle Constraints in Tabellen-Constraints übersetzt.

- Einen benannten Constraint entfernen:

```
ALTER TABLE STUDENTEN DROP CONSTRAINT SCH2
```

  Man benötigt zum Entfernen meist den Namen des Constraints. Man kann ihn im Data Dictionary nachsehen (USER_CONSTRAINTS).

ALTER TABLE (12)

Oracle (Constraints entfernen, fortgesetzt):

- Primär- und Alternativschlüssel kann man entfernen, ohne den Namen zu kennen:

```
ALTER TABLE STUDENTEN DROP PRIMARY KEY CASCADE
ALTER TABLE STUDENTEN DROP UNIQUE(VORNAME,NACHNAME)
```

- NOT NULL-Constraints können mit einer Spaltenmodifikation entfernt werden:

```
ALTER TABLE AUFGABEN MODIFY (THEMA NULL)
```

- Man kann Constraints auch aktivieren/deaktivieren.

  Eine deaktivierte Bedingung existiert noch im Data Dictionary, wird aber nicht überprüft/erzwungen.

ALTER TABLE (13)

Oracle (Tabellen umbenennen):

- Tabellen können in Oracle umbenannt werden. Dies geschieht mit einem speziellen "RENAME"-Befehl:

```
RENAME STUDENTEN TO STUD
```

- Dies ist im SQL-92-Standard nicht enthalten.

  Es funktioniert auch in DB2, aber nicht in SQL Server, MySQL oder Access. Oracle (aber nicht DB2) versteht auch "ALTER TABLE STUDENTEN RENAME TO STUD".

- Ich kenne keine Möglichkeit, Spalten oder DB-Nutzer in Oracle umzubenennen.

  Wenn Sie es wissen, sagen Sie es mir.

ALTER TABLE (14)

DB2 (Spalten hinzufügen):

- Das Hinzufügen von Spalten funktioniert wie im SQL-Standard:

```
ALTER TABLE STUDENTEN
      ADD COLUMN ZUSATZPKT NUMERIC(4,1)
```

- Ist ein Default-Wert festgelegt, kann die neue Spalte NOT NULL sein.
- Es gibt keine Möglichkeit, Spalten zu löschen oder umzubenennen.

ALTER TABLE (15)

DB2 (Spalten verändern):

- Die einzige erlaubte Modifikation einer Spalte ist die Änderung der Länge einer VARCHAR-Spalte:

```
ALTER TABLE AUFGABEN
      ALTER THEMA SET DATA TYPE VARCHAR(100)
```

  Es sind wieder nur Erhöhungen der Länge möglich.
- Es scheint so, dass der NULL/NOT NULL-Status nicht geändert werden kann.

  Man kann NOT NULL mit einem CHECK-Constraint schreiben, aber das System versteht die Äquivalenz nicht richtig, z.B. verlangt PRIMARY KEY die Bedingung NOT NULL.

ALTER TABLE (16)

DB2 (Constraints hinzufügen/entfernen):

- Hinzufügen funktioniert wie im SQL-Standard:

```
ALTER TABLE STUDENTEN
      ADD CHECK(ZUSATZPKT IS NOT NULL)
```

- Ein benannter Constraint kann entfernt werden (wie in SQL-92, aber ohne RESTRICT/CASCADE):

```
ALTER TABLE STUDENTEN
      DROP CONSTRAINT ZUSATZPKT_DEF
```

- Primärschlüssel kann man ohne Namen entfernen:

```
ALTER TABLE STUDENTEN DROP PRIMARY KEY
```


ALTER TABLE (17)

SQL Server (Spalten hinzufügen/löschen):

- Das Hinzufügen funktioniert wie im Standard, aber das Wort "COLUMN" nach "ADD" ist nicht erlaubt.

```
ALTER TABLE STUDENTEN ADD ZUSATZPKT NUMERIC(4,1)
```

  Ist ein Default-Wert festgelegt, kann die neue Spalte NOT NULL sein.
- Eine Spalte kann wie folgt gelöscht werden (das Schlüsselwort "COLUMN" wird verlangt, RESTRICT oder CASCADE werden nicht verstanden):

```
ALTER TABLE STUDENTEN DROP COLUMN ZUSATZPKT
```

ALTER TABLE (18)

SQL Server (Spalten verändern):

- Der Datentyp einer Spalte kann verändert werden, aber nicht, wenn die Spalte ein Schlüssel ist oder ein Check-Constraint vorliegt:

```
ALTER TABLE AUFGABEN
        ALTER COLUMN THEMA VARCHAR(100)
```

Verringerungen der Größe sind möglich, wenn die Daten noch passen.

- Man kann auch die NULL/NOT NULL-Bedingung in Spalten ändern:

```
ALTER TABLE AUFGABEN
        ALTER COLUMN THEMA VARCHAR(100) NULL
```


ALTER TABLE (19)

SQL Server (Constraints hinzufügen/entfernen):

- Das Hinzufügen funktioniert wie im SQL-Standard:

```
ALTER TABLE STUDENTEN
        ADD CHECK(ZUSATZPKT IS NOT NULL)
```

Eine Spalte muss Not Null sein, um einen Primärschlüssel-Constraint hinzuzufügen.

- Eine benannte Bedingung kann entfernt werden:

```
ALTER TABLE STUDENTEN
        DROP CONSTRAINT ZUSATZPKT_DEF
```

(Wie in SQL-92, aber ohne RESTRICT/CASCADE).

ALTER TABLE (20)

Access:

- Das Hinzufügen und Löschen von Constraints funktioniert wie im SQL-92-Standard.

  Das Schlüsselwort COLUMN ist optional, auch wenn es im Handbuch anscheinend verlangt wird. CASCADE und RESTRICT werden im Handbuch beim Löschen von Constraints nicht erwähnt, aber sie werden akzeptiert (obwohl sie wie im Standard nicht unbedingt nötig sind).

- Der Datentyp einer Spalte kann vielseitig geändert werden. Man kann z.B. zwischen Zahlen und Zeichenketten konvertieren (wenn die Zeichenketten ein numerisches Format haben).

```
ALTER TABLE STUDENTEN ALTER COLUMN ZUSATZPKT CHAR(20)
```


ALTER TABLE (21)

MySQL:

- MySQL hat ein sehr umfangreiches ALTER TABLE-Statement mit vielen Optionen (siehe Handbuch).

  Dies liegt zum Teil daran, dass MySQL ALTER TABLE ausführt, indem eine Kopie der gesamten Tabelle mit der neuen Struktur erstellt wird. Das kann bei großen Tabellen aufwändig werden und andere Systeme versuchen, dies zu vermeiden (was zu vielen Einschränkungen führt).

- Die SQL-92-Syntax wird verstanden, außer beim Löschen von benannten Constraints.

  Sogar RESTRICT/CASCADE werden zumindest syntaktisch für DROP COLUMN akzeptiert, auch wenn sie im Handbuch nicht aufgeführt sind.

ALTER TABLE (22)

MySQL, fortgesetzt:

- Es gibt eine nicht standardisierte Syntax für das Löschen der Constraints, die MySQL unterstützt.

  Mit "ALTER TABLE T DROP PRIMARY KEY" kann man den Primärschlüssel entfernen. Bei anderen Schlüsseln muss man den Namen X herausfinden, den MySQL dem Schlüssel bzw. Index zugeordnet hat (mit "SHOW CREATE TABLE T"), dann "ALTER TABLE T DROP INDEX X". CHECK-Constraints und Fremdschlüssel werden nicht unterstützt. Der Datentyp einer Spalte und der NULL/NOT NULL-Status werden wie folgt geändert: "ALTER TABLE AUFGABEN MODIFY THEMA VARCHAR(100) NULL".

- Tabellen und Spalten können umbenannt werden.

```
ALTER TABLE STUDENTEN RENAME TO STUD
ALTER TABLE STUDENTEN CHANGE SID STUD_NR NUMERIC(3) PRIMARY KEY
```