Part 9: Table Definition

References:

- Elmasri/Navathe:Fundamentals of Database Systems, 3rd Edition, 1999. Chap. 8, "SQL — The Relational Database Standard"
- Kemper/Eickler: Datenbanksysteme (in German), 4th Ed., Oldenbourg, 1997. Chapter 4: Relationale Anfragesprachen (Relational Query Languages).
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.
- Date/Darwen: A Guide to the SQL Standard, Fourth Edition, Addison-Wesley, 1997.
- van der Lans: SQL, Der ISO-Standard (in German), Hanser, 1990.
- Melton/Simon: Understanding the New SQL. Morgan Kaufman, 1993.
- Oracle 8i SQL Reference, Release 2 (8.1.6), Dec. 1999, Part No. A76989-01.
- Oracle 8i Concepts, Release 2 (8.1.6), Dec. 1999, Part No. 76965-01. Chapter 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, 2nd Edition (Part of MSDN Library Visual Studio 6.0). Microsoft Access 2000 Online Help.
- DuBois: MySQL. New Riders Publishing, 2000, ISBN 0-7357-0921-1, 756 pages.
- MySQL Reference Manual for Version 3.23.53.

Objectives

After completing this chapter, you should be able to:

- write `CREATE TABLE` commands in SQL.
- enumerate the general kinds of data types to expect in a DBMS.

  You should be able to explain the parameters of `NUMERIC`, `CHAR`, `VARCHAR`.
- define not null, key, foreign key, and check constraints.

Overview

Data Types (1)

- Each column can store only values of a specific data type (defined in the `CREATE TABLE` statement).
- The relational model does not depend on a specific selection of data types.
- Different DBMS products support a different selection of data types, but strings and numbers of different lengths and precision are always available.
- Modern systems allow user-defined data types.

  DB2, Oracle, and SQL Server support user-defined types.

Data Types (2)

- Data types define not only the set of possible values, but also operations (functions) on the values.
- In the SQL-86 standard, the only data type functions available were `+`, `-`, `*`, `/`.

  These functions can be expected in any DBMS.
- Other functions differ from DBMS to DBMS.

  So using them may lead to portability problems.
- E.g. the string concatenation operator `||` is contained in the SQL-92 standard, but SQL Server and Access use "`+`" instead, and MySQL "`concat(...)`".

Data Types (3)

Categories of Data Types:

- Relatively standardized:
  - ◇ Character strings (fixed length, variable length)
  - ◇ Numbers (integer, fixed point, floating point)
- Supported, but differently in each DBMS:
  - ◇ Long character data
  - ◇ Binary data
  - ◇ National language character strings
  - ◇ Date and time values
- Plus user-defined and DBMS-specific data types.

Character Strings (1)

`CHARACTER`(n):

- Fixed length strings of n characters.
- Data which is stored in a column of this type is filled with spaces to the defined length n.

  So disk space for n characters is always needed. If the length of the actual data varies significantly, one should use `VARCHAR`, see below.
- `CHARACTER`(n) can be abbreviated to `CHAR`(n).
- If no length is specified, 1 is assumed. Thus, "`CHAR`" (without length) allows storage of single characters.

  In Access, `CHAR` without length seems to be treated like `CHAR(255)`.

Character Strings (2)

- The data type `CHAR`(n) was already contained in the SQL-86 standard.

  Of course, all five systems (Oracle, SQL Server, DB2, Access, and MySQL) support it.
- The systems differ in the maximal possible value for the string length n.

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

Character Strings (3)

VARCHAR(n):

- Variable length string of up to n characters.

  So only space for the actual character data is needed.
  The maximal length n serves as a constraint, but normally does not influence the file/record format on the disk.

- This data type was added in the SQL-92 standard.
  It was not contained in the SQL-86 standard.
- However, probably all modern DBMS support it.

  It is supported in all five DBMS examined in this course (Oracle, DB2, SQL Server, Access, MySQL).

Character Strings (4)

- Officially, the type is called CHARACTER VARYING(n), but the standard allows the abbreviation VARCHAR.
- The systems differ in the maximal possible value for the maximal string length n:

DBMS	Maximal n
Oracle 8.0	4000
DB2 UDB 5	254/4000
SQL Server 7	8000
Access	255
MySQL	255

In DB2, if n is greater than 254, no sorts are possible for this column (includes ORDER BY, GROUP BY, DISTINCT).

String Functions (1)

String Functions 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).

String Functions (2)

String functions in the SQL-92 standard:

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

  CHARACTER_LENGTH(s) can be abbreviated to CHAR_LENGTH(s).

- 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).

String Functions (3)

String functions 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).

String Functions (4)

String functions in DB2, Continued:

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

String Functions (5)

String Functions in SQL Server:

- s_1+s_2 (concatenation).
- `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).

String Functions (6)

String Functions in SQL Server, continued:

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

  `REPLACE` replaces all occurrences of x in s by y. `STUFF` replaces characters with position n to $n+m$ in s by 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).

  This converts a number into a string. l is the output length.

String Functions (7)

String Functions 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).

String Functions (8)

String Functions in MySQL, Continued:

- 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).

String Functions (9)

String Functions in MySQL, Continued:

- 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).

String Functions (10)

String Functions 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($x, \ldots$).

String Functions (11)

String Functions in Access, Continued:

- 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).

String Functions (12)

String Functions in Access, Continued:

- 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).

Numbers (1)

- NUMERIC(p,s): Signed number with p digits in total, of which s are after the decimal point.

  This is also called a fixed point number, since the decimal point is always at the same position (in contrast to floating point numbers).

- E.g. NUMERIC(3,1) has values -99.9 to 99.9.

  MySQL permits values -99.9 to 999.9 (wrong).

- NUMERIC(p): Signed whole number of p digits.

  Whole numbers are also called "integers". NUMERIC(p) is the same as NUMERIC(p,0). "NUMERIC" without p uses an implementation-defined p.

- "NUMERIC(p[, s])" was already contained in SQL-86.

  It is not supported in Access, but in the other four DBMS.

Numbers (2)

- DECIMAL(p,s): Essentially the same as NUMERIC(p,s).

  Here the DBMS can choose a larger set of values. E.g. for type NUMERIC(1), the DBMS must print an error when one tries to insert 10, for DECIMAL(1), it may accept and store the value (if it anyway uses a whole byte for the attribute). By the way, MySQL never prints an error, it silently uses the maximal possible value.

- DECIMAL can be abbreviated "DEC".
- As NUMERIC, DECIMAL was already part of SQL-86.
- Oracle normally uses NUMBER(p,s) and NUMBER(p), but understands NUMERIC/DECIMAL as synonyms.

  None of the other four systems understands NUMBER.

Numbers (3)

- The precision p (total number of digits) can range from 1 to some maximum.

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

In SQL Server, the server must be started with the option `/p` to support up to 38 digits. Otherwise the limit is 28 digits. MySQL stores `NUMERIC(`p`,`s`)` as string of p digits plus characters for "-", ".". But MySQL does arithmetics with `DOUBLE` (about 15 digits precision).

- The scale s must satisfy $s \geq 0$ and $s \leq p$.

In Oracle, $-84 \leq s \leq 127$ (no matter what p is).

Numbers (4)

- `INTEGER`: Signed whole numbers, stored decimal or binary, range of values is implementation-defined.

DB2, SQL Server, MySQL and Access use 32 bit binary numbers: -2147483648 (-2^{31}) .. +2147483647 $(2^{31}-1)$. I.e. the range of values in these systems is slightly larger than `NUMERIC(9)`, but the SQL standard does not guarantee this. In Oracle: Synonym for `NUMBER(38)`.

- `INT`: Abbreviation for `INTEGER`.
- `SMALLINT`: As above, range of values may be smaller.

DB2, SQL Server, MySQL and Access use 16 bit binary numbers: -32768 (-2^{15}) .. +32767 $(2^{15}-1)$. Thus the range in these systems is larger than `NUMERIC(4)`, but smaller than `NUMERIC(5)`.
In Oracle, it is again a synonym for `NUMBER(38)`.

Numbers (5)

- Additional non-standard integer types:
  - `BIT`: In SQL Server (0,1), Access (-1, 0).

    In MySQL, `BIT` and `BOOL` are treated as synonyms for `CHAR(1)`.
  - `TINYINT`: In MySQL (−128 .. 127), in SQL Server (0 .. 255).

    In Access, the type `BYTE` can store values 0 .. 255.
  - `BIGINT`: In DB2 and MySQL $(-2^{63} \,..\, 2^{63}-1)$.

    The range is larger than `NUMERIC(18)`.
  - MySQL supports also e.g. `INTEGER UNSIGNED`.

    In MySQL, one can also specify a display width, e.g. `INTEGER(5)`, and add the keyword `ZEROFILL` to specify that e.g. 3 is displayed e.g. as 0003 if the display width is 4.

Numbers (6)

- `FLOAT(`p`)`: Floating point number $M * 10^E$ with at least p bits of precision for M $(-1 < M < +1)$.
- `REAL` and `DOUBLE PRECISION` are abbreviations for `FLOAT(`p`)` with implementation defined values for p.
- E.g. SQL Server (DB2 and MySQL are similar):
  - `FLOAT(`p`)`, $1 \leq p \leq 24$, uses 4 Bytes.

    7 digits precision (range `-3.40E+38` to `3.40E+38`).
    `REAL` means `FLOAT(24)`.
  - `FLOAT(`p`)`, $25 \leq p \leq 53$, uses 8 Bytes.

    15 digits precision (range `-1.79E+308` to `+1.79E+308`).
    `DOUBLE PRECISION` means `FLOAT(53)`.

Numbers (7)

- Oracle uses `NUMBER` (without parameters) as datatype for floating point numbers.

  Oracle also understands `FLOAT`(p). `NUMBER` allows storage of values between $1.0 * 10^{-130}$ and $9.9 \ldots * 10^{125}$ with 38 decimal digits of precision.

- Access understands `REAL`, `FLOAT` (without parameter), and `DOUBLE` (without `PRECISION`).
- Whereas `NUMERIC`, `DECIMAL` etc. are exact numeric data types, `FLOAT` is an approximate numeric type: Rounding errors are not really controllable.

  E.g. for money, never use `FLOAT`.

Numbers (8)

Operations for Numbers 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).

Numbers (9)

Operations for Numbers in the SQL-92 Standard:

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

Operations for Numbers 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).

Numbers (10)

Operations for Numbers in DB2, continued:

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

Numbers (11)

Operations for Numbers in SQL Server:

- x + y, x - y, x * y, x / y, x % y (modulo), - x, + x.
- x & y, x | y, x ^ y, ~ x (bit operations).
- 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($x, n, 1$) (truncates).

Numbers (12)

Operations for Numbers in SQL Server, continued:

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

Numbers (13)

Operations for Numbers 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.

Numbers (14)

Operations for Numbers in MySQL, Continued:

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

  There are restrictions for using RAND.

Numbers (15)

Operations for Numbers 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).

  In addition, there are functions for computing interest payments etc.

Data Types in SQL-86

- CHAR[ACTER][(n)] [...] marks optional parts.
- NUMERIC[(p[,s])]
- DEC[IMAL][(p[,s])]
- INT[EGER], SMALLINT
- FLOAT[(p)], REAL, DOUBLE PRECISION
- These types should be very portable.

  Four systems (Oracle, DB2, SQL Server. MySQL) understand them. Access does not spport NUMERIC, DECIMAL, FLOAT(p), DOUBLE PRECISION. All five systems support VARCHAR, which was not contained in SQL-86.

Overview

1. Classical SQL Data Types
2. More SQL Data Types
3. CREATE TABLE Syntax
4. CREATE SCHEMA, DROP TABLE
5. ALTER TABLE

Long Character Data (1)

Oracle:

- LONG: Character strings of up to 2GB length.
- The use of LONG columns is very restricted. Basically, they only allow to store a file inside the database, and retrieve it, but not use it in conditions or computations in SQL.

  E.g. LIKE, ||, LENGTH and other string functions cannot be used for LONG values. A table can have at most one column of type LONG. The input/output of LONG values is possible in the usual way, e.g. via SELECT-lists. In SQL*Plus, SET LONG n defines the maximal output length.

Long Character Data (2)

Oracle, continued:

- If a longer text has to be searchable with `LIKE`, it must be split into lines/paragraphs, stored separately as `VARCHAR` values.
- `CLOB`: Character large object, up to 4GB.

  This is something like a file, stored inside the database, with its own identity (LOB locator). It is very similar to `LONG`.
- `CLOB` is new in Oracle8. Oracle7 had only `LONG`.

  Probably, `LONG` is supported only for backwards compatibility.

Long Character Data (3)

Oracle, continued:

- Differences of `CLOB` to `LONG` are, e.g.:
  - ◇ The programming interface allows random access to the data in a `CLOB`.

    `LONG` values can read only sequentially.
  - ◇ A table can contain multiple `CLOB` columns.
  - ◇ `LOB` values can be stored in another tablespace (disk, file) than the table in which they occur.
- `CLOB` values can not be used in conditions, except via the PL/SQL procedures in the package `DBMS_LOB`.

Long Character Data (4)

DB2:

- Also DB2 has character large objects (up to 2GB).

  Character large objects are declared with a maximal size, e.g. `CLOB(1M)`, which influences the length of the descriptor used to point to the actual data. The actual data is stored separately from the table rows.
- Already `VARCHAR` columns of length > 254 cannot be used in e.g. `ORDER BY`, `GROUP BY`, `DISTINCT`.

  Everything that requires sorting is excluded.
- In addition, `CLOB(`n`)` columns cannot be used with `=`, `<>`, `<`, `<=`, `>`, `>=`, `IN`, `BETWEEN`.

Long Character Data (5)

DB2, continued:

- However, `CLOB(`n`)` columns can be used with `LIKE`.
- There is also a type `LONG VARCHAR` (up to 32700 bytes) which is retained for backward compatibility.

Long Character Data (6)

SQL Server:

- SQL Server has a data type "TEXT" which can store up to 2GB of character data.

  Actually, the maximum size is $2^{31} - 1$, i.e. 2 147 483 647.

- TEXT columns can be used in the WHERE clause only with LIKE or IS NULL.

  E.g. =, <>, <, <=, >, >=, IN, BETWEEN cannot be used.

- TEXT columns cannot be used with DISTINCT, ORDER BY, GROUP BY.

Long Character Data (7)

SQL Server, continued:

- TEXT columns are also not allowed as arguments to + (string concatenation), but there are some data-type functions such as DATALENGTH, SUBSTRING which work with TEXT.

Long Character Data (8)

MySQL:

- BLOB: "Binary large object" of up to 64 KByte.

  The exact maximum size is $2^{16} - 1 = 65\,535$ Bytes.

- TEXT: Character string of maximal length 64 KByte.

  The only difference between BLOB and TEXT is that comparisons are case-sensitive for BLOB and case-insensitive for TEXT.

- MEDIUMBLOB/MEDIUMTEXT: Max. 16 MB.

  The exact maximum length is $2^{24} - 1 = 16\,777\,215$.

- LONGBLOB/LONGTEXT: Max. 4 GB.

  The exact maximum length is theoretically $2^{32} - 1 = 4\,294\,967\,295$. However,the current version of MySQL has a limit of 16 MB.

Long Character Data (9)

MySQL, Continued:

- In general, BLOB and TEXT are treated like VARCHAR(n) with a large n.

  However, MySQL removes trainling spaces when VARCHAR-values are stored, but it does not do so for BLOB and TEXT.

- Sorting (GROUP BY, ORDER BY, DISTINCT) looks only at the first 1024 Bytes.

  This limit can be increased with the parameter max_sort_length. From Version 3.23.2, one can have indexes on BLOB and TEXT columns.

- It is recommended to use the SUBSTRING-function to extract a value that can be sorted.

Long Character Data (10)

Access:

- `MEMO`: Long Text Data.

  The maximum size is 65.535 characters when entering data through the user interface and 1 GB when entering data via the program interface. (Actually, many different values are mentioned in the manuals: 64.000 characters, 65.535 characters, 1.2 GB, and 2.14 GB.) `LONGTEXT` is a synonym for this type.

- `OLEOBJECT`: E.g. Microsoft Word Document.

  Maximal size is 1 GB. `LONGBINARY` is a synonym for this type. One cannot use `DISTINCT`, `GROUP BY`, or `ORDER BY` for values of this type. The manual lists the same restrictions for `MEMO` fields, but there these constructs seemed to work (in Access 2000). `OLEOBJECT` values are displayed on output as "Long binary data".

Bit Strings in SQL-92

- `BIT(`n`)`: Bit strings of exactly n bits.

  Constants of this type are written either in binary, e.g. `B'11000101'` or in hexadecimal, e.g. `X'C5'`. There is also a type `BIT VARYING(`n`)`.

- Bit strings were not contained in SQL-86, and are supported in none of the five systems.

  However, each system has some provision for binary data, and SQL Server and Access even have a type `BIT` (without length).

- The SQL-92 standard has no boolean type.

  SQL Server has `BIT`, Access has `YESNO`. Oracle and DB2 have no specific support for boolean values. Normally `CHAR(1)` is used together with a constraint that only `'Y'` and `'N'` are allowed in this column. MySQL treats `BIT` and `BOOL` as synonyms for `CHAR(1)` (without constraint). In Access and MySQL, the boolean columns can be used as conditions.

Binary Data (1)

Oracle:

- `RAW(`n`)`: Binary data of length n bytes.

  n must be between 1 and 2000.

- Binary data can be used, e.g. for graphics.

  Whenever the meaning of the data is interpreted by an external program and not known to the database.

- Usually Oracle converts character data if the user (client, e.g. PC) has another character set than the server. This is not done for `RAW` data.

Binary Data (2)

Oracle, continued:

- `RAW` data are written in SQL statements as character strings with hexadecimal digits.
- `LONG RAW`: Variable length binary data, up to 2GB.
- `BLOB`: Binary large object, up to 4GB.

Binary Data (3)

DB2:

- String columns can be declared as containing binary data:

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

- This ensures that
  - ◇ comparison is done literally (binary codes),

    I.e. the collation sequence is not used which might e.g. identify uppercase and lowercase letters.
  - ◇ and no conversion is done between different code pages.

Binary Data (4)

DB2, continued:

- String constants can be written in hexadecimal notation, e.g. `X'FFFF'`.
- There are also binary large objects, `BLOB`(n), which can hold up to 2GB of binary data.

Binary Data (5)

SQL Server:

- `BINARY`(n): Fixed-length binary data of n bytes.

  The size n can be at most 8000. Input data is filled with `0x00` bytes to the declared length n.
- `VARBINARY`(n): Variable-length binary data.

  n is the maximal length in bytes. It can be at most 8000.
- Constants are written in the form `0xFF1C`.
- `IMAGE` can store up to 2GB of binary data.

  It is the BLOB type of SQL Server. Despite its name, SQL Server does not interpret the data in it, e.g. does not store a graphic format for the image (giff, jpeg, etc.).

Binary Data (6)

MySQL:

- `CHAR`(n) `BINARY`: Fixed length string of n bytes.

  The effect of adding `BINARY` to the string data type is that comparisons are done by comparing the byte codes for exact equality. Without `BINARY`, comparisons are done case-insensitive.
- `VARCHAR`(n) `BINARY`: Variable-length string, $\leq n$ bytes.
- `BLOB`, `MEDIUMBLOB`, `LONGBLOB`: see above.
- String constants can be written in hex: `0x612D7A`.

  Otherwise, when inserting binary data, the bytes 0 (ASCII NUL), 34 (ASCII `"`), 39 (ASCII `'`), 92 (ASCII `\`) must be encoded by using escape sequences (`\0`, `\"`, `\'`, `\\`).

Binary Data (7)

Access:

- BINARY, BINARY(n): Byte string.

  In Access 2000, n must be $\leq$ 510.

- Comparisons with this type are case-sensitive.
- OLEOBJECT, LONGBINARY: see above.
- Hexadecimal constants of the form 0x61002D007A00 are possible.

  Since Access uses Unicode, one must write two bytes for every character. The above example is the string 'a-z'. Note that the bytes of a 16-bit integer are "swapped". The condition 'a'=0x6100 evaluates to true.

Date/Time Types (1)

SQL-92:

- DATE: A value between 0001-01-01 (January 1st, 1 AD) and 9999-12-31 (December 31st, 9999 AD).

  Of course, illegal dates such as 1999-02-29 are excluded. DATE constants are written as a character string of the form YYYY-MM-DD, marked with the keyword DATE, e.g. DATE '1965-06-26'.

- TIME: Time of day (00:00:00 to 23:59:59).

  Seconds can have a fractional part, e.g. TIME(3) allows to store values like 16:20:31.001. A second part up to 61.9 is tolerated to allow leap seconds. TIME constants are written as a string of the form HH:MM:SS[.SSS], preceded by "TIME", e.g. TIME '09:30:00'.

- SQL-92 has also support for different time zones.

Date/Time Types (2)

SQL-92, continued:

- TIMESTAMP: DATE and TIME(6) together.

  E.g.: TIMESTAMP '1999-03-23 18:30:00.000000'.

- INTERVAL DAY(p): Period of time in days.

  Values are n days, where $-10^p < n < 10^p$. So INTERVAL DAY(3) is a difference between two DATE values (positive or negative), which cannot exceed 999 days. Constants are written as, e.g., INTERVAL '14' DAY.

- INTERVAL HOUR(p) TO SECOND: Difference between two TIME values in hours ($< 10^p$), minutes, and seconds.

  Constants are written, e.g.: "INTERVAL '2:12:35' HOUR TO SECOND".
  Instead of "HOUR TO SECOND" one can use, e.g. "DAY TO MINUTE"
  or whatever components/precision one would like.

Date/Time Types (3)

DB2:

- DB2 supports DATE, TIME, and TIMESTAMP.

  TIME is always in seconds, a precision cannot be specified. However, TIMESTAMP has microseconds as required. There are no specific constants for date and time values (e.g. TIME '09:30:00' is not understood), but character strings in a number of formats are automatically converted. 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 has no INTERVAL type.

  But certain intervals can be specified as arguments of + and -.
  E.g. DUE_DATE + 21 DAYS < CURRENT DATE works,
  but CURRENT DATE - DUE_DATE > 21 DAYS is illegal.

Date/Time Types (4)

Oracle:

- Oracle does not support the SQL-92 types.
- Oracle has a type for timestamps, called `DATE`.

  Despite its name, `DATE` also stores a time of day (in hours, minutes, seconds). If only a date is specified, Oracle assumes 00:00:00am (midnight, beginning of the day).

- There are no specific `DATE` constants, but Oracle does type conversion for strings automatically.

  Strings of the form `'DD-MON-YY'` (e.g. `'23-MAR-99'`) are accepted where date values are needed. Use `TO_DATE` and `TO_CHAR` for other formats (including times). The default format depends on `NLS_DATE_FORMAT`: In Germany, `'DD.MM.YY'` is used instead of `'DD-MON-YY'`.

Date/Time Types (5)

SQL Server:

- `DATETIME`: From Jan 1, 1753 to Dec 31, 9999.

  Date and time (like Oracle's `DATE`). Accuracy: 0.003s.

- `SMALLDATETIME`: From Jan 1, 1900 to June 6, 2079.

  Accuracy: 1 minute. Needs 4 bytes (`DATETIME`: 8 bytes).

- There are no specific `DATETIME` constants, but string constants are automatically transformed.

  The default output format is `'2000-03-29 18:00:00'`. However, SQL Server understands also other formats, e.g. `'March 29, 2000'` (if the time is missing, `00:00` is assumed), `'29-MAR-2000 12:00'`, `'03/27/00 9:00 PM'`, `'14:30:00'` (if the date is missing, `01.01.1900` is assumed).

Date/Time Types (6)

MySQL:

- `DATETIME`: Date and time (accurate to seconds).

  Range `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`. The usual format for constants is `'YYYY-MM-DD HH:MM:SS'`. Some alternative formats are understood (but the year must come first). MySQL permits also illegal dates like `'2002-02-31 00:00:00'`, and especially year, month, day can be zero (which gives a kind of partial null value).

- `DATE`: Date (Range `'1000-01-01'` to `'9999-12-31'`).
- `TIME`: Time of day and time interval.

  The range is `-838:59:59` to `838:59:59` (more than 34 days backward to 34 days forward). One has to include seconds in `TIME`-constants, e.g. `'06:15'` is understood as `'00:06:15'`. Formats are, e.g. `'HH:MM:SS'`, `'HHH:MM:SS'`, `'DD HH:MM:SS'` (`DD` are days between 0 and 33).

Date/Time Types (7)

MySQL, Continued:

- `YEAR`

  The range is 1901 to 2155 (1 Byte). The usual format is `'YYYY'`. Two-digit years from 70 to 99 are converted to 1970 to 1999, and from 00 to 69 are understood as 2000 to 2069.

- `TIMESTAMP`: Current date and time (in seconds).

  Values range from 1970 to sometime in 2037. MySQL treats a column of type `TIMESTAMP` special: It has as default value automatically the current date and time (if a table has several `TIMESTAMP` columns, this only applies to the first column). Thus, if the application does not specify a value for this column on insertion, the date and time when the tuple was created is stored in the `TIMESTAMP` column. Timestamp values are displayed as an integer, e.g. in the format `YYYYMMDDHHMMSS`. One can specify a display size, e.g. `TIMESTAMP(8)`: `YYYYMMSS`.

Date/Time Types (8)

Access:

- DATETIME: Date between years 100 and 9999 plus time (accurate to seconds).

  Stored as floating point value: Integer part is the number of days since December 30, 1899. Fractional part is the number of seconds since midnight.
  Probably years before 100 are excluded in order to be able to detect a two-digit specification of the year.

- Constants are written e.g. in the form #MM-DD-YYYY# or #MM-DD-YYYY HH:MM:SS#.

  One can also use "/" instead of "-" or write the year with two digits.

National Languages (1)

- Oracle can store special German characters like ä, ö, ü, ß, and it can also work e.g. with Japanese characters.

  Oracle transforms characters between different encoding schemes, e.g. in a client/server environment.

- Oracle can print error messages etc. in different languages.

  Also things like the correct sorting sequence and the format for date values etc. can be adapted to national needs.

- Some decisions such as the DB character set have to be made when the database is installed.

National Languages (2)

SQL-92:

- The SQL-92 standard also contains a large section on national character sets (which is different from Oracle).

SQL Server:

- A code page for CHAR, VARCHAR, and TEXT can be selected during the installation of SQL server.
- NCHAR, NVARCHAR, and NTEXT store strings in Unicode (2 bytes per character). Constants e.g. N'aäb'

National Languages (3)

DB2:

- The CREATE DATABASE statement has options CODESET and TERRITORY.
- In this way a code page (possibly double byte) is determined.
- Types GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, and DBCLOB store double byte characters.

Access:

- Access uses Unicode for text types.

Other Data Types (1)

Oracle:

- `BFILE`: A Reference to an Operating System File.

  The file itself is not stored inside the DB, only its name. In contrast to `CLOB` and `BLOB`, the transaction management does not apply. External files are read-only for the DB.

- `ROWID`: Physical pointer to a specific row.

  The `ROWID` specifies file, block, and tuple number. Accesses via the `ROWID` are especially fast. Every table has a "pseudo-column" `ROWID` (it can be used like a real column under `SELECT` and `WHERE`). `ROWID` components are shown with e.g. `DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)`.

- User-defined PL/SQL data types.

Other Data Types (2)

SQL Server:

- `MONEY` and `SMALLMONEY`: Numbers with an accuracy of 1/10000 of a monetary unit (i.e. scale = 4).

  `SMALLMONEY` are 32-bit numbers from `-214748.3648` to `+214748.3647`, `MONEY` are 64-bit (up to 922 billion). Constants are written `$12.34`.

- `TIMESTAMP`: DB-wide unique number, automatically updated.

- `UNIQUEIDENTIFIER`: Globally unique identifier.

- `CURSOR`: Reference to a cursor.

  This is an SQL query (possibly in execution) or query result.

Other Data Types (3)

DB2:

- `DATALINK`: Reference to a file stored outside the DB (URL).

Access:

- `CURRENCY`: Number with 4 digits after decimal point.

  −922 337 203 685 477.5808 to 922 337 203 685 477.5807 (64 Bit number). Displayed with currency symbol (e.g. `$10.25`).

- `GUID`: System-generated unique number.

  16 Bytes. One should not write to columns of this type.

Other Data Types (4)

MySQL:

- `ENUM`$(v_1, v_2, \ldots)$: One of the values v_i.

  The values are written as string constants, e.g. `ENUM('MON','TUE',...)`. An enumeration type can have up to 65 535 distinct values. If an invalid value (outside the set $\{v_1, v_2, \ldots\}$) is inserted, MySQL silently inserts the empty string instead (as an error value). Values are numbered from 1, the empty string has the number 0. MySQL permits comparisons with numbers, one can also apply the usual operations for numbers (e.g. `+`). MySQL sorts enumeration type values by their numeric value, i.e. in the order in which they were declared.

- `SET`$(v_1, v_2, \ldots)$: Any subset of $\{v_1, v_2, \ldots\}$.

  Again, the values are written as string constants. A set can have at most 64 members. Set constants are written as a string with the element names separated by commas (the v_i cannot contain commas).

Overview

Example (1)

STUDENTS			
<u>SID</u>	FIRST	LAST	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

EXERCISES			
<u>CAT</u>	<u>ENO</u>	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

RESULTS			
<u>SID</u>	<u>CAT</u>	<u>ENO</u>	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

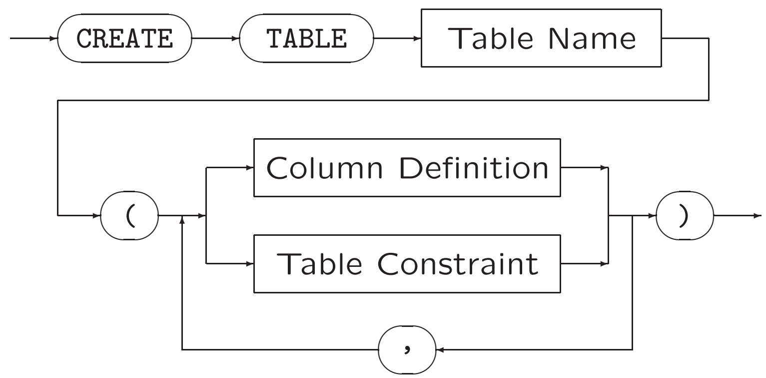
Example (2)

- The CREATE TABLE statement in SQL defines:
  - Table name
  - Columns and their data types
  - Constraints (keys, foreign keys, NOT NULL, CHECK)
- E.g. STUDENTS(<u>SID</u>, FIRST, LAST, EMAIL^{o}):

```
CREATE TABLE STUDENTS (
       SID   NUMERIC(3)  NOT NULL CHECK(SID > 0)
                         PRIMARY KEY,
       FIRST VARCHAR(20) NOT NULL,
       LAST  VARCHAR(20) NOT NULL,
       EMAIL VARCHAR(128),
       UNIQUE(FIRST, LAST) )
```

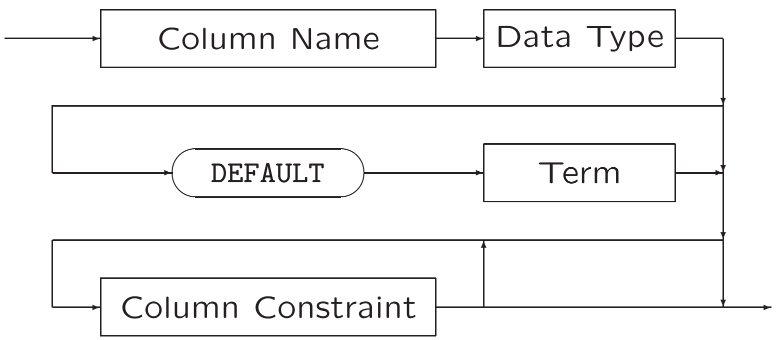

CREATE TABLE: Syntax (1)

CREATE TABLE Statement:



CREATE TABLE: Syntax (2)

Column Definition:



Constraints: Overview (1)

- In SQL, constraints can be defined as “column constraints” or “table constraints”.
- Column constraints are constraints that refer to only one column. Table constrains can refer to more than one column.
- Column constraints (except possibly `NOT NULL`) can also be formulated as “table constraints”, but the syntax of column constraints is slightly simpler.

Internally, column constrains are translated to table constraints.

Constraints: Overview (2)

- Column constraints are specified in the column definition (separated only by a space).
- Column constraints in the example:

```
CREATE TABLE STUDENTS (
       SID   NUMERIC(3)   NOT NULL  CHECK(SID>0)
                          PRIMARY KEY,
       FIRST VARCHAR(20)  NOT NULL,
       LAST  VARCHAR(20)  NOT NULL,
       EMAIL VARCHAR(128),
       UNIQUE(FIRST, LAST) )
```


Constraints: Overview (3)

- Table constraints are separated by a comma from each other and the column definitions:
- Table constraint in the example:

```
CREATE TABLE STUDENTS (
       SID   NUMERIC(3)  NOT NULL CHECK(SID>0)
                         PRIMARY KEY,
       FIRST VARCHAR(20) NOT NULL,
       LAST  VARCHAR(20) NOT NULL,
       EMAIL VARCHAR(128),
       UNIQUE(FIRST, LAST) )
```

Constraints: Overview (4)

- The same example with the column constraints (except `NOT NULL`) replaced by table constraints:

```
CREATE TABLE STUDENTS (
       SID   NUMERIC(3)  NOT NULL,
       FIRST VARCHAR(20) NOT NULL,
       LAST  VARCHAR(20) NOT NULL,
       EMAIL VARCHAR(128),
       PRIMARY KEY (SID),
       CHECK(SID > 0),
       UNIQUE(FIRST, LAST) )
```


Column Constraints (1)

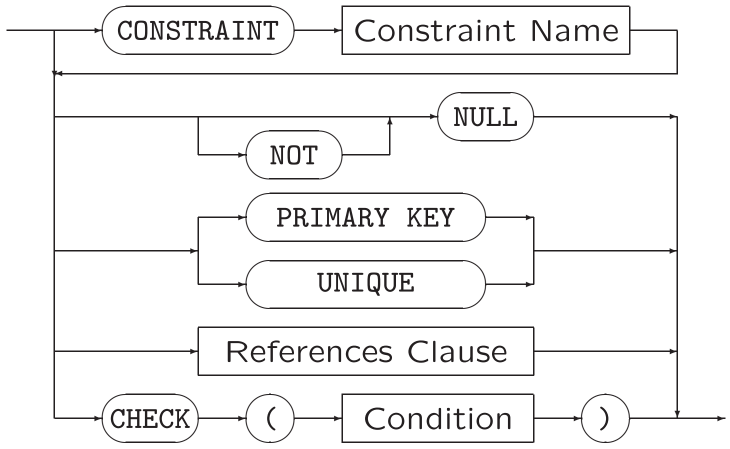
- There are five kinds of column constraints:
  - ◇ `NOT NULL`: Null values are excluded in this column.
  - ◇ `PRIMARY KEY`: This column is the primary key of the table.
  - ◇ `UNIQUE`: This column is an alternative key.
  - ◇ `REFERENCES` T: This column is a foreign key.

    I.e. values in this column must appear in the primary key column of the table T.
  - ◇ `CHECK` (C): Values in this column must satisfy C.

    C is a condition like in the `WHERE`-clause, but with certain restrictions (see below).

Column Constraints (2)

Column Constraint:



Column Constraints (3)

- One can write "`NULL`" as a column constraint in order to emphasize that null values are allowed.

  However, this is not really a constraint and it is anyway the default.
- `PRIMARY KEY` implies `NOT NULL`.

  However, DB2 requires that `NOT NULL` is explicitly specified in addition. In MySQL, `NOT NULL` must be explicitly specified if the primary key is declared as table constraint.
- There can be only one primary key per table.
- `UNIQUE` does not imply `NOT NULL`.

  In DB2, `UNIQUE` can only be used together with `NOT NULL`.

Column Constraints (4)

- In SQL-86, only [`NOT NULL` [`UNIQUE`]] was supported.
- Furthermore, `UNIQUE` was not implemented in many systems.
- In older code "`CREATE UNIQUE INDEX`" was often used to enforce key constraints.

  See Part 14 about physical design.
- In 1989, the standard was extended by an optional "Integrity Enhancement Feature" (SQL-89).

  This contained the above constructs.

CHECK Constraints (1)

- The condition C of `CHECK`-constraints looks like a `WHERE`-condition without subqueries. Also functions like `SYSDATE` that can change later are excluded.

  In column constraints, only this column can be accessed. Otherwise use a table constraint (see below).
- The basic idea of efficient constraint checking is that the constraint was satisfied before the update.
- The DBMS checks only the inserted/modified row.

  Since such constraints hold in the empty DB state, and the system ensures that updates do not destroy their validity, it follows by induction that they hold in every database state. This also explains why `SYSDATE` is excluded: Constraints could become false without update.

CHECK Constraints (2)

- `CHECK`-constraints are not supported in MySQL and Access.
- Oracle, SQL Server, and DB2 all exclude subqueries in `CHECK`-constraints.

  Thus, the basic restriction `CHECK`-constraints is that they must be evaluable for each single tuple in isolation.
- SQL-92 permits subqueries in `CHECK`-constraints, but then constraint checking is difficult/inefficient.

  If a table referenced in the subquery is changed, it is in general difficult to find out for which rows the `CHECK`-condition must be evaluated again. A simple solution would be to check all rows, but then each small update causes a long integrity check.

CHECK Constraints (3)

- If subqueries inside `CHECK` were implemented, a foreign key could be formulated in this way:

Not Implemented!

```
CREATE TABLE RESULTS(
   SID NUMERIC(3)
       CHECK(SID IN (SELECT SID FROM STUDENTS)),
   ... )
```

- Here it is clear that if a `STUDENTS` tuple t is deleted or its `SID` is updated, only tuples in `RESULTS` can be affected that have the (old) SID of t.

  Thus, in this case it is not necessary to evaluate the `CHECK`-constraint again for all tuples in `RESULTS`.

Constraints and Null Values

- Integrity constraints count as valid if they result in the truth value "unknown"

  The definitions for keys and foreign keys which consist of multiple attributes, of which only some are null is actually complicated and system-dependent.

- E.g. with this declaration, the email address can be null, but if it is not, it must contain "@":

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


Constraint Names (1)

- A constraint can optionally be given a name by prefixing it with "`CONSTRAINT` ⟨`Name`⟩".

  MySQL accepts constraint names only for table constraints. However, it seems that it anyway immediately forgets them.

- Constraint names must be unique in the schema, whereas column names only have to be unique within a table.

  In DB2, constraint names must be unique only in a table.
  In DB2, `NOT NULL` constraints cannot be named.

- Always define a name (except for `NOT NULL`). Otherwise the system chooses names like "`SYS_C036`".

Constraint Names (2)

- When the constraint is violated, its name is printed: System-generated names give bad error messages.

  The error message for a not null constraint is usually clear:
  `ORA-01400: cannot insert NULL into (USER.TABLE.COLUMN)`
  If there are several keys, this is already unclear:
  `ORA-00001: unique constraint (BRASS.SYS_C007916) violated`
  For a foreign key, one gets the following message:
  `ORA-02291: integrity constraint (BRASS.SYS_C007914) violated - parent key not found.`
  For a check constraint, this message:
  `ORA-02290: check constraint (BRASS.SYS_C007915) violated.`

- Dropping a constraint later is easier if its name is known.

Constraint Names (3)

- Example with explicit constraint names:

```
CREATE TABLE STUDENTS (
       SID   NUMERIC(3) NOT NULL
             CONSTRAINT SID_MUST_BE_POSITIVE
                        CHECK(SID > 0)
             CONSTRAINT STUDENTS_KEY
                        PRIMARY KEY,
       FIRST VARCHAR(20) NOT NULL,
       LAST  VARCHAR(20) NOT NULL,
       EMAIL VARCHAR(128),
       CONSTRAINT STUDENT_NAMES_MUST_BE_UNIQUE
                  UNIQUE(FIRST, LAST) )
```

Table Constraints (1)

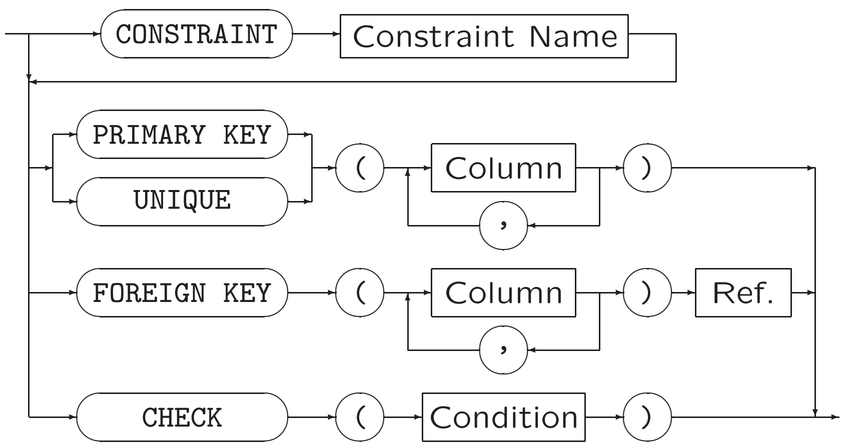
- Table constraints are needed when a key or a foreign key consists of more than one attribute, or a CHECK-condition refers to more than one attribute.

  Constraints referring only to one column can be defined as either table constraints or column constraints.

- The four kinds of table constraints are:
  - ◇ PRIMARY KEY(A_1, ..., A_2)
  - ◇ UNIQUE(A_1, ..., A_2)
  - ◇ FOREIGN KEY(A_1, ..., A_2) REFERENCES R
  - ◇ CHECK(C)

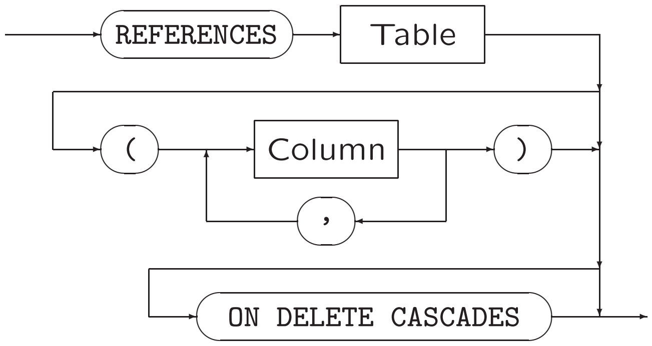
Table Constraints (2)

Table Constraint:



Foreign Keys (1)

References Clause (Ref.):



Foreign Keys (2)

- The References Clause is used after the column name in column constraints or after the FOREIGN KEY specification in table constraints.
- It is possible to specify the referenced column names, e.g. REFERENCES STUDENTS(SID).

  However, only a key (PRIMARY KEY or UNIQUE) can be referenced. If the columns are not mentioned, the primary key of the table is assumed. Referencing an alternate key seldom makes sense.

- The foreign key and the referenced key must consist of the same number of columns and corresponding columns must have the same data type.

Foreign Keys (3)

- MySQL does not check foreign keys, but it accepts foreign key declarations in `CREATE TABLE` statements.
- Consider the FK `RESULTS(SID`$\rightarrow$`STUDENTS, ...)`.
- One can request (in SQL-92, Oracle, DB2, not in SQL Server or Access) that if a student is deleted, his/her homework results are deleted, too:

  `REFERENCES STUDENTS ON DELETE CASCADES`
- Otherwise the system rejects attempts to delete a student who still has results in the DB.

Foreign Keys (4)

- Consider a database with course information: `COURSES(..., INSTRUCTOR`^{o}$\rightarrow$`FACULTY, ...)`.
- In SQL-92 and DB2 (not in Oracle, SQL Server, and Access), one can specify that if a faculty member is deleted, his/her courses remain in the DB, but their attribute `INSTRUCTOR` is set to a null value.
- This is written: "`ON DELETE SET NULL`".
- In SQL-92, one can also choose "`SET DEFAULT`". This is not supported in any of the five systems.

Foreign Keys (5)

- In SQL-92, one can also specify the effects of updates on the key of `FACULTY` (name change).
- This is written "`ON UPDATE ...`". It is not supported in any of the three systems.

  DB2 understands "`ON UPDATE`" with the parameters "`NO ACTION`" and "`RESTRICT`", but rejecting updates of referenced key values is the default.

Default Column Values (1)

- In the command for creating new rows (`INSERT`), users do not have to define values for all columns.

  `INSERT` is explained in Part 11. If a row is inserted via a view, the user may not even be able to specify values for all columns.
- Usually, a null value is put in the missing columns.
- However, it is possible to specify a default value which should be stored in columns for which no value was given.
- One can also specify that a column value must be explicitly defined: "`NOT NULL`" and no `DEFAULT`.

Default Column Values (2)

- E.g. exercises should have MAXPT = 10 if the INSERT-command does not specify a value for MAXPT:

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

- "DEFAULT SYSDATE" puts the current date into the column (the date when the row was inserted).

  "SYSDATE" works only in Oracle. In SQL Server and SQL-92, use "CURRENT_TIMESTAMP". In DB2, write "CURRENT TIMESTAMP". MySQL automatically uses the current date and time as default value for the first column of type "TIMESTAMP" (one cannot explicitly declare this.).

Default Column Values (3)

- Several users may be allowed to insert rows into the same table. With "DEFAULT USER" the name of the current user is stored in a column (e.g. ENTERED_BY).

  In this way, the user who performed the insertion can be logged. "USER" was already contained in the SQL-86 standard and should be highly portable. But MySQL and Access both do not support "USER".

- Database users can be given selective INSERT-rights, so that the user cannot specify values for certain columns.

  Then the user cannot override the DEFAULT-clause, thus the username or the current date are always stored as defined in the DEFAULT-clause.

Default Column Values (4)

- The DEFAULT clause was not contained in SQL-86, and Access does not support it.
- In SQL-92 the default value can only be
  - ◇ a constant,
  - ◇ a function without arguments, or

    More specifically: USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, CURRENT_DATE, CURRENT_TIME[(...)], CURRENT_TIMESTAMP[(...)]
  - ◇ NULL.
- Oracle accepts any term without column names.
- MySQL only permits constants as default values.

Default Column Values (5)

- "DEFAULT NULL" is used when no default value is explicitly declared.
- Thus, if a column is NOT NULL and no default value is specified, any attempt to insert a row without defining a value for this column gives an error.
- MySQL avoids this by assigning e.g. the empty string or 0 as default value for NOT NULL columns when no default value is explicitly declared.

  This violates the standard and makes error detection more difficult. One also cannot explicitly declare "DEFAULT NULL" for a NOT NULL column.

Unique Numbers (1)

- One often needs to generate unique numbers, e.g. for students, customers, invoices (primary key).
- SQL-92 does not contain a mechanism for that.
- Theoretically, it is not difficult to query the current maximum in the table, add one, and use the result.

  Alternatively, one can also create a table with only one row and one column which contains the current maximum. This might be slightly faster, although finding the maximum with a B-tree index is also fast.
- But with concurrent users this becomes difficult, see Part 11. Therefore, most DBMS have a special mechanism for generating unique numbers.

Unique Numbers (2)

Oracle:

- Oracle has database objects called "sequences":

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

- One can retrieve a number and increment the value stored in the sequence as follows:

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

  (DUAL is a dummy table with exactly one row.)
- One cannot declare STUD_IDS.NEXTVAL as DEFAULT, but one can use it in the INSERT-statement.

Unique Numbers (3)

SQL Server:

- In Microsoft SQL Server, one can add the keyword IDENTITY to the declaration of an integer column:

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

- Values for this column normally cannot be specified, the next number is automatically inserted.

  Even when no column list in the INSERT-statement is used, the column is excluded in the VALUES-list. But it is clearer to explicitly specify the columns. IDENTITY cannot be used together with DEFAULT.
- One can specify starting value and increment:

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


Unique Numbers (4)

DB2:

- In DB2 one can specify that a column value is computed e.g. by a unique number generator:

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

- "GENERATED ALWAYS": In the INSERT-statement, one cannot specify a value for this column.

  The alternative is GENERATED BY DEFAULT. Both cases exclude a DEFAULT clause. There are more parameters for the unique number generator, e.g. IDENTITY(START WITH 101, INCREMENT BY 1).

Unique Numbers (5)

MySQL:

- In MySQL, one can add the keyword `AUTO_INCREMENT` to the declaration of an integer column:

```
SID INT AUTO_INCREMENT PRIMARY KEY
```

- This works only with the binary integer types.

  E.g. it does not work with `NUMERIC(3)`. One can use `UNSIGNED` types. The column must be declared as a key (`PRIMARY KEY` or `UNIQUE`).

- If the `INSERT`-statement specifies `NULL` or `0` for this column, the next number is inserted instead.

  A declared default value is silently ignored.

Unique Numbers (6)

Access:

- Access has the column data type `COUNTER` that produces unique numbers:

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

- One can specify starting value and increment with parameters of this data type:

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

- In order to insert the next value of the counter, one must explicitly list the columns in the `INSERT`-statement, and leave the column `SID` out.

Temporary Tables

- The SQL-92 standard and some DB systems permit to declare temporary tables which are basically
  - ◇ automatically deleted at the end of the transaction or session,

    Depending on the system, it might be that the table itself is not deleted, only all rows in it are removed.

  - ◇ invisible to other (concurrent) sessions.

    Normally, tables of other users are invisible (unless the user granted the access rights). Two users can create tables with the same name. Here different database connections, even for the same user, have a different namespace.

- See the manual of your DBMS for details.

Storage Parameters

- In most DBMS, the "`CREATE TABLE`" statement has a long list of storage parameters which can be set.
- These parameters belong to the physical schema, not to the conceptual schema.

  It is a bit unfortunate that both things are mixed up here.

- The SQL standard does not contain any definition belonging to the physical schema.
- If performance is important, try to understand the meaning of these parameters and set them.

Restrictions (1)

SQL Server:

- max. 1024 columns per table
- max. 8060 bytes per row

  The data of TEXT etc. columns are stored separately.
- Keys can contain max. 16 columns.
- Key values can be max. 900 bytes.

  This limits basically the concatenation of the column values for all columns.

Restrictions (2)

DB2:

- max. 500 columns per table
- max. 4005 bytes per row

  Including the descriptors of LOB columns, but not their data.
- Keys can contain max. 16 columns.
- Key values can be max. 255 bytes.

Restrictions (3)

Oracle:

- max. 1000 columns per table.
- max. 32 columns per key.
- Key values (or indexed values) are limited to 40% of the block size.

  The database block size can be configured within reasonable limits (when the database is created).
- The row length is not limited, but the performance degrades if a row must be stored in multiple blocks.

Restrictions (4)

Access:

- max. 255 columns per table.
- max. 10 columns per key.
- max. 2000 characters in a row.

  Not counting Memo and OLE object columns.
- The maximal size of a table is 1 GB.
- max. 2 GB per database (.mdb file).

  One can add links to tables in other files to work around this limit.

Overview

1. Classical SQL Data Types
2. More SQL Data Types
3. CREATE TABLE Syntax
4. CREATE SCHEMA, DROP TABLE
5. ALTER TABLE

CREATE SCHEMA (1)

- In Oracle and SQL Server, DB accounts and schemas are 1:1.

  If some user needs more than one schema, multiple accounts for the same person must be created.

- Tables are identified system-wide by their name and owner.

  In SQL Server: server, database, owner, name.

- In DB2, schemas and users are separate things, although every schema belongs to exactly one user.

CREATE SCHEMA (2)

- There is a `CREATE SCHEMA` command, but it is only needed when two tables reference each other:

```
CREATE SCHEMA AUTHORIZATION ⟨Account⟩
   CREATE TABLE DEPT(DNO NUMERIC(2) PRIMARY KEY,
      HEAD NUMERIC(4) REFERENCES EMP, ...)
   CREATE TABLE EMP(EMPNO NUMERIC(4) PRIMARY KEY,
      DNO NUMERIC(2) REFERENCES DEPT, ...);
```

- Note that the single `CREATE TABLE` statements are not terminated with a “;” or in any other way.

CREATE SCHEMA (3)

- In all three systems, `CREATE SCHEMA` can contain `CREATE TABLE`, `CREATE VIEW` and `GRANT` commands.

  In DB2, also `COMMENT ON` and `CREATE INDEX` is allowed.

- If any of the statements fails, all are undone.
- Since DB2 permits several schemas per user, the syntax is there

  `CREATE SCHEMA` ⟨Name⟩ `AUTHORIZATION` ⟨User⟩

  The schema name and the authorization clause are both optional, but at least one of the two is required in DB2.

- MySQL and Access do not support `CREATE SCHEMA`.

Deleting Tables (1)

- Tables can be deleted with the command

  `DROP TABLE ⟨Table Name⟩`

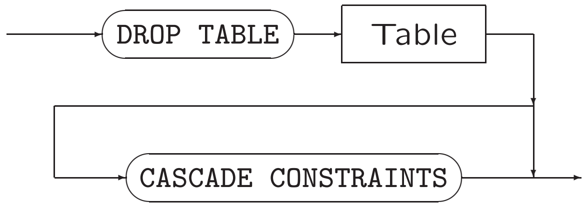
- Of course, this also deletes all rows in the table.

  Be careful! In Oracle, dropping a table automatically ends a transaction. So no undo is possible for this action.

Deleting Tables (2)

- What if the table is referenced in foreign key constraints?
  - ◇ In SQL Server and Access, the referencing table must be dropped first.
  - ◇ In DB2, the foreign key is automatically dropped.
  - ◇ In Oracle, "`CASCADE CONSTRAINTS`" can be specified to drop the foreign keys, too.
  - ◇ In SQL-92, it is only "`CASCADE`".

Deleting Tables (3)



Examples:

- `DROP TABLE STUDENTS CASCADE CONSTRAINTS`
- `DROP TABLE RESULTS`

  If one drops first the table `RESULTS` (containing the foreign key) and then `STUDENTS`, no `CASCADE CONSTRAINTS` is necessary.

Overview

1. Classical SQL Data Types
2. More SQL Data Types
3. CREATE TABLE Syntax
4. CREATE SCHEMA, DROP TABLE
5. ALTER TABLE

ALTER TABLE (1)

- The purpose of the `ALTER TABLE` command is to modify the schema of an existing table.
- In principle, one could copy the data to a temporary location, drop the table, create the table again with the modified schema, and copy the data back. But:
  - ◇ Copying the data is impractical for large tables.
  - ◇ Table entries may be referenced in foreign keys: One might end up recreating the entire DB.

    Also indexes, grants, views, triggers, stored procedures etc. reference tables. Some of this information will be lost when the table is dropped.

ALTER TABLE (2)

- Examples for table schema changes:
  - ◇ New columns can be added to a table.

    Because columns can be added, it is safer to specify columns in application programs instead of `SELECT *`.
  - ◇ The width of existing columns can be increased.

    E.g. from `VARCHAR(20)` to `VARCHAR(30)`.
    This is actually not possible in the SQL-92 standard, but in all three DBMS (Oracle, DB2, SQL Server).
  - ◇ Constraints can be removed or added.

    Some systems also have the possibility to disable a constraint, so that it is no longer checked, but still stored in the system. It can then later be enabled again.

ALTER TABLE (3)

- `ALTER TABLE` was not contained in SQL-86.
- It is contained in the SQL-92 standard, but concrete DBMS implementations differ quite heavily in the syntax and in what exactly can be changed.
- The SQL-92 standard offers these possibilities:
  - ◇ Columns can be added to or dropped from tables.
  - ◇ The default column value can be changed, but the data type cannot.
  - ◇ Constraints can be added or removed.

ALTER TABLE (4)

SQL-92 (Adding Columns):

- E.g. add a column "`EXTRA_PT`" to "`STUDENTS`":

```
ALTER TABLE STUDENTS
      ADD COLUMN EXTRA_PT NUMERIC(4,1)
                          CHECK(EXTRA_PT >= 0)
```

- The keyword "`COLUMN`" is optional.
- The new column will first be null for all existing rows.
- It can then be updated to some other value.

  It might, however, create efficiency problems if the rows become much longer than they were when the were inserted.

ALTER TABLE (5)

SQL-92 (Adding Columns, Continued):

- If a default value is specified, the new column can be NOT NULL:

```
ALTER TABLE STUDENTS
   ADD EXTRA_PT NUMERIC(4,1) DEFAULT 0 NOT NULL
```

- The new column is added as the last (rightmost) column of the table.

  There is no way to add it at any other position.

- Views (stored queries) are not affected, because SELECT * is replaced by by an explicit column list when the view definition is processed.

ALTER TABLE (6)

SQL-92 (Removing Columns):

- A column can be removed from a table:

```
ALTER TABLE STUDENTS
      DROP COLUMN EXTRA_PT RESTRICT
```

- RESTRICT means that the ALTER TABLE fails if the column is referenced in a constraint or a view.

  Column constraints for this column (or table constraints that reference only this column) do not count: They are automatically deleted.

- The alternative is CASCADE: The referencing database object is deleted, too.

  There is no default: One must specify RESTRICT or CASCADE.

ALTER TABLE (7)

SQL-92 (Modifying Columns):

- The only allowed modification of a column is to change its default value:

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

- The default can be set to null also with this syntax:

```
ALTER TABLE EXERCISES
ALTER COLUMN MAXPT DROP DEFAULT
```

- In SQL-92, it is not possible to change the data type of a column.

ALTER TABLE (8)

SQL-92 (Modifying Constraints):

- Add a constraint:

```
ALTER TABLE STUDENTS
      ADD CONSTRAINT EXTRA_DEFINED
      CHECK(EXTRA_PT IS NOT NULL)
```

  Only table constraints can be added, but column constraints are anyway only syntactic shorthands.

- Remove a named constraint:

```
ALTER TABLE STUDENTS
      DROP CONSTRAINT EXTRA_DEFINED RESTRICT
```

  Date/Darwen state that the standard requires to add RESTRICT or CASCADE although this makes sense only for keys referenced in foreign keys (CASCADE means to remove those foreign keys, too).

ALTER TABLE (9)

Oracle (Adding Columns):

- E.g. add a column "EXTRA_PT" to "STUDENTS":

```
ALTER TABLE STUDENTS ADD
  (EXTRA_PT NUMERIC(4,1) CHECK(EXTRA_PT >= 0))
```

- For existing rows, the new column will be null.

  Of course, one can update the rows to set another value.

- If a default value is specified, the new column can be NOT NULL:

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


ALTER TABLE (10)

Oracle (Modifying Columns):

- The datatype of a column can be modified:

```
ALTER TABLE EXERCISES MODIFY (TOPIC VARCHAR(100))
```

- The width of a column cannot be decreased.

  Unless it contains only null values.

- The MODIFY clause cannot be used for changing column constraints except NULL/NOT NULL.

  There is another syntax for this, see next slide.

- In Oracle, columns cannot be dropped or renamed.

ALTER TABLE (11)

Oracle (Adding/Dropping Constraints):

- Add a constraint:

```
ALTER TABLE STUDENTS ADD
    (CONSTRAINT ALTKEY UNIQUE(FIRST, LAST))
```

- The syntax for table constraints must be used here.

  Internally, all column constraints are anyway translated into table constraints.

- Drop a named constraint:

```
ALTER TABLE STUDENTS DROP CONSTRAINT ALTKEY
```

  Normally the constraint name is needed for dropping an existing constraint. If one does not know it, one can look it up in the data dictionary (table USER_CONSTRAINTS).

ALTER TABLE (12)

Oracle (Dropping Constraints, Continued):

- Primary and alternative keys can be dropped without knowing their name:

```
ALTER TABLE STUDENTS DROP PRIMARY KEY CASCADE
ALTER TABLE STUDENTS DROP UNIQUE(FIRST, LAST)
```

- NOT NULL constraints can be removed with a column modification:

```
ALTER TABLE EXERCISES MODIFY (TOPIC NULL)
```

- In Oracle, constraints can also be enabled/disabled.

  A disabled constraint still exists in the data dictionary, but is not checked/enforced.

ALTER TABLE (13)

Oracle (Renaming Tables):

- Tables can be renamed in Oracle. This is done with a special "RENAME" command:

```
RENAME STUDENTS TO STUD
```

- This is not contained in the SQL-92 standard.

  It works also in DB2, but not in SQL Server, MySQL, or Access. Oracle also understands "ALTER TABLE STUDENTS RENAME TO STUD", but DB2 does not.

- I do not know of any way to rename columns or database users in Oracle.

  If you know, please tell me.

ALTER TABLE (14)

DB2 (Adding Columns):

- Adding a column is done as in the SQL standard:

```
ALTER TABLE STUDENTS
      ADD COLUMN EXTRA_PT NUMERIC(4,1)
```

- If a default value is specified, the column can be not null.
- There is no way to drop or rename a column.

ALTER TABLE (15)

DB2 (Modifying Columns):

- The only modification of a column is to change the size of a VARCHAR-column:

```
ALTER TABLE EXERCISES
      ALTER TOPIC SET DATA TYPE VARCHAR(100)
```

  Again, only increases in size are possible.

- It seems that the NULL/NOT NULL status cannot be changed.

  You can write NOT NULL with a CHECK-constraint, but the system doesn't really understand the equivalence, e.g. PRIMARY KEY requires NOT NULL.

ALTER TABLE (16)

DB2 (Adding/Removing Constraints):

- A constraint can be added as in the SQL standard:

```
ALTER TABLE STUDENTS
      ADD CHECK(EXTRA_PT IS NOT NULL)
```

- A named constraint can be dropped (basically as in SQL-92, but without RESTRICT/CASCADE):

```
ALTER TABLE STUDENTS
      DROP CONSTRAINT EXTRA_DEFINED
```

- A primary key can be dropped even without name:

```
ALTER TABLE STUDENTS DROP PRIMARY KEY
```

ALTER TABLE (17)

SQL Server (Adding/Removing Columns):

- Adding a column is done as in the standard, but the keyword "COLUMN" after "ADD" is not allowed.

  ```
  ALTER TABLE STUDENTS ADD EXTRA_PT NUMERIC(4,1)
  ```

  If a default value is specified, the column can be not null.

- A column can be dropped as follows (the keyword "COLUMN" is required, and RESTRICT/CASCADE are not understood):

  ```
  ALTER TABLE STUDENTS DROP COLUMN EXTRA_PT
  ```


ALTER TABLE (18)

SQL Server (Modifying Columns):

- The data type of a column can be altered, but not if this column has a key or check constraint:

  ```
  ALTER TABLE EXERCISES
      ALTER COLUMN TOPIC VARCHAR(100)
  ```

  Decreases in size are possible if the data still fits.

- One can also change the NULL/NOT NULL requirement for columns:

  ```
  ALTER TABLE EXERCISES
      ALTER COLUMN TOPIC VARCHAR(100) NULL
  ```


ALTER TABLE (19)

SQL Server (Adding/Removing Constraints):

- Constraints can be added as in the SQL standard:

  ```
  ALTER TABLE STUDENTS
      ADD CHECK(EXTRA_PT IS NOT NULL)
  ```

  A column must be not null to add a primary key constraint.

- A named constraint can be dropped:

  ```
  ALTER TABLE STUDENTS
      DROP CONSTRAINT EXTRA_DEFINED
  ```

  (As in SQL92, but without RESTRICT/CASCADE.)

ALTER TABLE (20)

Access:

- Adding and removing columns as well as constraints is done as in the SQL-92 standard.

  It works with and without the optional keyword COLUMN, although the grammar in the manual seems to specify that it is required. The grammar does not mention CASCADE and RESTRICT for dropping constraints, but they are accepted (although not required as in the standard).

- The data type of a column can be changed in very generous ways, e.g. one convert between numbers and strings (if the strings have numeric format).

  ```
  ALTER TABLE STUDENTS ALTER COLUMN EXTRA_PT CHAR(20)
  ```

ALTER TABLE (21)

MySQL:

- MySQL has a very rich ALTER TABLE statement with many options (see the manual).

  In part this is because MySQL executes the ALTER TABLE by creating a copy of the entire table with the new structure. This can be expensive for large tables and other systems try to avoid this (which then results in many restrictions).

- The SQL-92 syntax is understood, except for dropping named constraints.

  Even the keywords RESTRICT/CASCADE are at least syntactically accepted for the DROP COLUMN, although they are not listed in the manual.

ALTER TABLE (22)

MySQL, continued:

- There is a non-standard syntax for dropping the constraints that are supported by MySQL.

  One can use "ALTER TABLE T DROP PRIMARY KEY" to remove the primary key. For other keys one must use "SHOW CREATE TABLE T" to find out the name X assigned to the key/index by MySQL (e.g. the column name) and then use "ALTER TABLE T DROP INDEX X". CHECK-constraints and foreign keys are anyway not supported. The data type of a column and its NULL/NOT NULL-status can be changed with a syntax similar to Oracle: "ALTER TABLE EXERCISES MODIFY TOPIC VARCHAR(100) NULL".

- Tables and columns can be renamed.

```
ALTER TABLE STUDENTS RENAME TO STUD
ALTER TABLE STUDENTS CHANGE SID STUD_NO NUMERIC(3) PRIMARY KEY
```