Part 11: SQL III

Elmasri/Navathe:

- Chap. 8, "SQL — The Relational Database Standard"

I have also used the following sources:

- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Edition, McGraw-Hill, 1999. Chapter 4: "SQL", Section 6.2: "Referential Integrity".
- 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, there is an English version), Hanser, 1990.
- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- Michael Gertz: Oracle/SQL Tutorial, 1999. [`http://www.db.cs.ucdavis.edu/teaching/sqltutorial/`]
- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- Oracle8 SQL Reference, Oracle Corporation, 1997, Part No. A58225-01.
- Oracle8 Concepts, Release 8.0, Oracle Corporation, 1997, Part No. A58227-01.
- Don Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data.

Objectives

- You should know what general kinds of data types you can expect in a DBMS.
- You should understand that data types are one area where DBMS differ in many details (you must look into the manual).
- You should know the `VARCHAR` and `NUMERIC` data types in some detail.
- You should be able to write a `CREATE TABLE` statement, including keys, foreign keys, and `CHECK` constraints.
- You should be able to modify the database state with `INSERT`, `UPDATE`, and `DELETE`.

Overview

1. Data Types
2. Table Definition
3. Update Commands
4. ALTER TABLE

Data Types (1)

- The relational model does not depend on a specific selection of data types.
- Different DBMS products support a quite different selection of data types, although strings and numbers of different lengths and precision are always available.
- Modern systems even allow you to define your own data types.

  DB2, Oracle, and SQL Server have some support for user-defined types.
- Data types define not only the set of possible values, but also which operations are available on the types.

  E.g. $+$, $-$, $*$, $/$ for numbers, `||` for strings.

  See Appendix H for lists of data type operations.

Data Types (2)

Categories of Data Types:

- Relatively standarized:
  - Character strings (fixed length, variable length)
  - Numbers (integer, fixed point, floating point)
- Supported in one way or another, but different 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 the space for n characters is always needed.
- CHARACTER can be abbreviated to CHAR.
  If no length is specified, 1 is assumed.

  So the data type "CHAR" allows to store single characters.
- This data type was already contained in the SQL-86 standard. Of course, Oracle, SQL Server, DB2 support it.

  In Oracle, the size n must be between 1 and 2000.
  The maximum size is 8000 in SQL server, and 254 in DB2.

Character Strings (2)

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.
- Officially, the type is called CHARACTER VARYING(n), but the SQL-92 standard allows the abbreviation VARCHAR.
- VARCHAR is supported in Oracle, SQL Server, DB2.

  Oracle: $1 \leq n \leq 4000$. SQL Server: $1 \leq n \leq 8000$.
  DB2: $1 \leq n \leq 4000$, but if $n > 254$, no sort is possible.

Character String Comparisons (1)

- The outcome of string comparisons (=, <>, <, <=, >, >=) may depend on the DBMS or settings within the DBMS.
- The SQL-92 standard defines the notion of "collation sequences" (or short "collations") which determine
  - for any pair X and Y of characters, whether $X < Y$, $X = Y$, or $X > Y$, and
  - whether the blank-padded semantics (PAD SPACE) or the non-padded semantics (NO PAD) is used.
- You can expect that 'a' < 'b' etc., and that 'A' < 'B' and so on. But already comparisons between uppercase and lowercase characters are not so clear.

Character String Comparisons (2)

- The default installation/settings in the three DBMS are:
  - In Oracle all uppercase characters come before all lowercase characters, e.g. `'Z' < 'a'`.
  - In DB2, uppercase and lowercase characters are interleaved, e.g.: `'a' < 'A'`, `'A' < 'b'`.

    This also means that `'ac' < 'Ab'`.
  - SQL Server is by default case-insensitive, e.g.: `'a' = 'A'`.
- It might be possible to change this, but e.g. only during installation (SQL Server), or during database creation (Oracle, DB2).

Character String Comparisons (3)

- When the order (<, =, >) of every two characters is known, the comparison of strings of the same length is clear:
  - The system compares character by character, the first comparison which does not give “=” determines the result.
- For strings of different lengths, there are
  - “Non-Padded Comparison Semantics”: E.g. `'a' < 'a '`.

    Strings are compared character by character. When one string ends and no difference was found, the shorter string is considered less than the longer one.
  - “Blank-Padded Comparison Semantics”: E.g. `'a' = 'a '`.

    The shorter string is filled with `' '` before the comparison.

Character String Comparisons (4)

- DB2 and SQL Server seem to use always the blank-padded semantics (at least by default).
- In Oracle, the nonpadded semantics is used if at least one operand of a comparison operator has type `VARCHAR2`.

  Oracle has introduced a type `VARCHAR2(`n`)`. It is currently synonymous to `VARCHAR(`n`)`, but Oracle intends to change the comparison semantics for `VARCHAR`, while the semantics for `VARCHAR2` will remain stable. String literals (constants) in the query have type `CHAR(`n`)`. E.g. you can successfully compare `CHAR(10)` and `CHAR(20)` columns, and also these columns with, e.g., `'abc'`. But `CHAR(10)` and `VARCHAR(20)` can only be equal if the `VARCHAR` happens to be of length 10.

Numbers (1)

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

  E.g. `NUMERIC(3,1)` would allow to store `99.9`, `-12.3`.
- `NUMERIC(`p`)`: Signed whole number (integer) of p digits.

  I.e. `NUMERIC(`p`)` is an abbreviation for `NUMERIC(`p`,0)`.
  “`NUMERIC`” without p uses an implementation-defined p.
- `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 you want to insert 10, for `DECIMAL(1)`, it may accept and store the value (if it anyway uses a whole byte for the attribute).

  `DECIMAL` can be abbreviated “`DEC`”.

Numbers (2)

- The precision p can range from 1 to some maximum.

  The maximum is 38 in Oracle, 31 in DB2, and 28 in SQL Server (or 38 if the server was started with the option /p).
- The scale s must satisfy $s \geq 0$ and $s \leq p$.

  This is required in SQL-92, DB2, SQL Server.
  In Oracle, $-84 \leq s \leq 127$ (no matter what p is). Negative scales generate zeros at the end, scales which are greater than the precision generate zeros after the decimal point before the stored digits.
- Oracle normally uses `NUMBER(`p`,`s`)` and `NUMBER(`p`)`, but understands `NUMERIC(...)` and `DECIMAL(...)` as synonyms.

  Neither DB2 nor SQL Server understand `NUMBER`.

Numbers (3)

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

  DB2 and SQL Server use 32 bit binary numbers:
  -2 147 483 648 (-2^{31}) .. +2 147 483 647 ($2^{31}-1$).
  Oracle treats `INTEGER` as a synonym for `NUMBER(38)`.

  `INTEGER` can be abbreviated “`INT`”.
- `SMALLINT`: As `INTEGER`, but possibly smaller range of values.

  DB2 and SQL Server use 16 bit binary numbers:
  -32 768 (-2^{15}) .. +32 767 ($2^{15}-1$).
  Oracle treats it again as a synonym for `NUMBER(38)`.
- SQL Server has in addition `TINYINT` (0 .. 255) and `BIT` (0, 1), DB2 has `BIGINT` (-2^{63} .. $2^{63}-1$).

Numbers (4)

- `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 shorthands for `FLOAT(`p`)` with implementation defined values for p.
- E.g. SQL Server (DB2 is similar):
  - `FLOAT(`p`)`, $1 \leq p \leq 24$, uses 4 Bytes and gives 7 digits precision (range `-3.40E+38` to `3.40E+38`).
  - `FLOAT(`p`)`, $25 \leq p \leq 53$ uses 8 Bytes and gives 15 digits precision, (range `-1.79E+308` to `+1.79E+308`).
  - `REAL` is a shorthand for `FLOAT(24)` (i.e. 4 Bytes), `DOUBLE PRECISION` for `FLOAT(53)` (i.e. 8 Bytes).

Numbers (5)

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

  `NUMBER` allows to store values between $1.0 * 10^{-130}$ and $9.9\ldots * 10^{125}$ with 38 decimal digits of precision.
- Whereas `NUMERIC`, `DECIMAL`, `INTEGER`, `SMALLINT` are exact numeric data types, `FLOAT` is an approximate numeric type:
  - Rounding errors are not really controllable.
    Floating point values are always a bit “fuzzy”.
  - E.g. for money, you should never use `FLOAT`.

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. All three systems (Oracle, DB2, SQL Server) understand them (plus VARCHAR).

Long Character Data in Oracle (1)

- 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. You can input/output LONG values normally, e.g. use them in SELECT-lists. In SQL*Plus, use SET LONG n to define the maximal output length.
- If you want a longer text to be searchable with LIKE, you must split it into lines/paragraphs, stored separately as VARCHAR values.

Long Character Data in Oracle (2)

- CLOB: Character large object, up to 4GB (new in Oracle8).

  This is something like a file, stored inside the database, with its own identity (LOB locator). It is very similar to LONG.
- Differences 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 more than one column of type CLOB.
  - LOB values can be stored in another tablespace/file than the table in which they occur (“out of line” storage).
- CLOB values can still not be used in conditions, except via the PL/SQL procedures from the package DBMS_LOB.

Long Character Data in DB2

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

  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 (requires sorting).
- In addition, CLOB(n) columns cannot be used with =, <>, <, <=, >, >=, IN, BETWEEN.
- However, CLOB(n) columns can be used with LIKE.
- There is also a type LONG VARCHAR (up to 32 700 bytes) which is retained for backward compatibility.

Long Character Data in 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. you cannot use =, <>, <, <=, >, >=, IN, BETWEEN.
- You cannot use TEXT columns with DISTINCT, ORDER BY, GROUP BY.
- You cannot use TEXT columns as arguments to + (string concatenation), but there are some datatype functions such as DATALENGTH, SUBSTRING which work with TEXT.

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’.
- BIT VARYING(n): Bit strings of up to n bits.
- Bit strings were not contained in the SQL-86 standard, and are supported in none of the three systems.
  However, each system has some provision for binary data, and SQL Server even has a type BIT (without length).
- Astonishingly, neither the SQL-92 standard nor one of the three DBMS has a boolean data type.
  You would normally use CHAR(1) together with a constraint that only ’Y’ and ’N’ are allowed in this column.

Binary Data in 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) has another character set than the database (e.g. between PC and DB server). This is not done for RAW data.
- Input/Output as character strings with hexadecimal digits.
- LONG RAW: Variable length binary data, up to 2GB.
- BLOB: Binary large object, up to 4GB.

Binary Data in DB2

- You can declare string columns 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.
- You can write string constants 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 in 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 of up to n bytes.
  The size n 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.).

Date/Time Types in SQL-92 (1)

- `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 `DATE`, e.g. `DATE '1965-06-26'`.
- `TIME`: A time of day (between 00:00:00 and 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 in SQL-92 (2)

- `TIMESTAMP`: `DATE` and `TIME(6)` together.
  E.g.: `TIMESTAMP '1999-03-23 18:30:00.000000'`.
- `INTERVAL DAY(`p`)`: Period of time, n days, $-10^p < n < 10^p$.
  So `INTERVAL DAY(3)` is a difference between two `DATE` values (positive or negative), which cannot exceed 999 days. Constants of this type are, e.g., “`INTERVAL '14' DAY`”.
- `INTERVAL HOUR(`p`) TO SECOND`: Difference between two `TIME` values in hours ($< 10^p$), minutes, and seconds.
  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 in DB2

- DB2 supports `DATE`, `TIME`, and `TIMESTAMP`.
  `TIME` is always in seconds, you cannot specify a precision. 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 you can specify certain intervals 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 in Oracle

- Oracle does not support the SQL-92 date/time types.
- Oracle has a type for timestamps, called `DATE`.
  Despite its name, `DATE` also stores a time of day (in hours, minutes, seconds). If you specify only a date, Oracle assumes 00:00:00am (midnight, beginning of the day).
- Oracle has no specific constants for the data type `DATE`, but it 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` (see Appendix H) 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 in SQL Server

- SQL Server does not support the SQL-92 date/time types.
- `DATETIME`: From January 1, 1753 to December 31, 9999.
  Date and time (like Oracle's `DATE`). Accuracy: 0.003s.
- `SMALLDATETIME`: From January 1, 1900 to June 6, 2079
  Accuracy: 1 minute. Needs 4 bytes (`DATETIME`: 8 bytes).
- There are no specific `DATETIME` constants, but string constants in various formats 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).

National Language Character Sets (1)

- Oracle can store special German characters like ä, ö, ü, ß, and it can also work e.g. with Japanese characters.
  It can transform 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.
- The SQL-92 standard also contains a large section on national character sets (which is different from Oracle).

National Language Character Sets (2)

SQL Server:

- You can select a code page for `CHAR`, `VARCHAR`, and `TEXT` when you install SQL server.
- `NCHAR`, `NVARCHAR`, and `NTEXT` store strings in Unicode (2 bytes per character). Constants e.g. N'aäb'

DB2:

- The `CREATE DATABASE` statement has options `CODESET` and `TERRITORY`.
- In this way you determine a code page, which can be double byte. Types `GRAPHIC`, `VARGRAPHIC`, `LONG VARGRAPHIC`, and `DBCLOB` store double byte characters.

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` (you can use it like a real column under `SELECT` and `WHERE`). To see the `ROWID` components, use 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 $1/10\,000$ of a monetary unit (i.e. 4 digits after the “`.`”).
  `SMALLMONEY` are 32-bit numbers from `-214 748.3648` to `+214 748.3647`, `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 (SQL query or query result).

DB2:

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

Overview

1. Data Types
2. Table Definition
3. Update Commands
4. ALTER TABLE

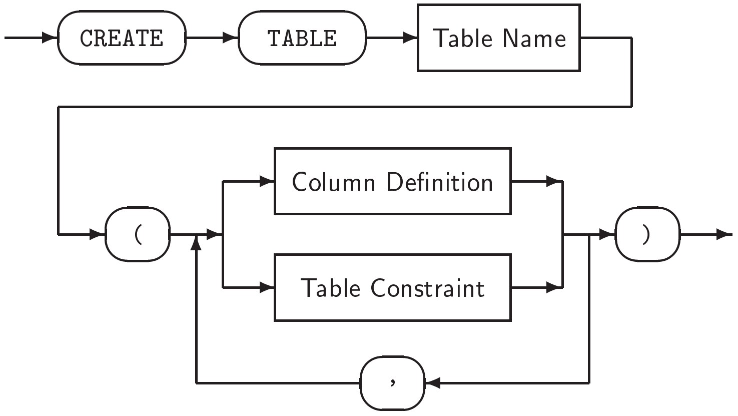
Example

- Course(<u>CRN</u>, Title, IName^{o}→Instructor)
- In SQL:

```
CREATE TABLE Course(
      CRN   NUMERIC(5)   NOT NULL CHECK(CRN > 0),
      Title VARCHAR(40)  NOT NULL,
      IName VARCHAR(20),
      PRIMARY KEY (CRN),
      FOREIGN KEY (IName) REFERENCES Instructor)
```

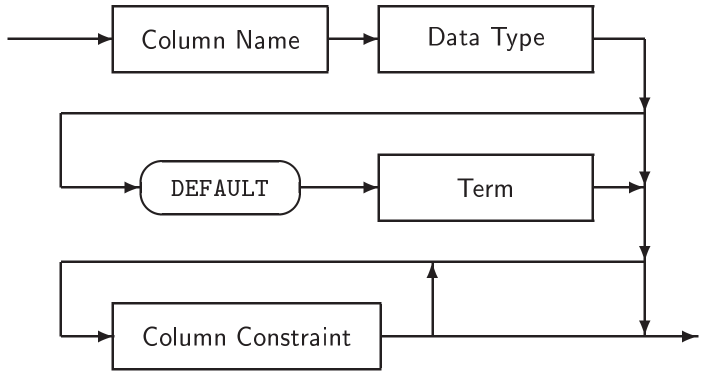
- The referenced table “`Instructor`” must already exist, unless both `CREATE TABLE` statements are contained together in a `CREATE SCHEMA` definition.

CREATE TABLE: Syntax (1)



CREATE TABLE: Syntax (2)

Column Definition:



Default Values for Columns (1)

- In the command for creating new table rows (see below), users do not have to define values for all columns.
- Usually, a null value is put in the missing columns.
- However, you can explicitly specify a default value which should be stored in columns for which no value was given.
- E.g. if a new instructor is stored, and no phone number is given, one can automatically insert the department number:

```
CREATE TABLE Instructor(
        IName   VARCHAR(20)   PRIMARY KEY,
        Phone   NUMERIC(7)    DEFAULT 6249400)
```


Default Values for Columns (2)

- With “`DEFAULT SYSDATE`” one can put the current date into the column, with “`DEFAULT USER`” the current user.

  “`USER`” was already contained in the SQL-86 standard and should be highly portable. “`SYSDATE`” works only in Oracle. In SQL Server and SQL-92, use “`CURRENT_TIMESTAMP`”. In DB2, you must write “`CURRENT TIMESTAMP`”.
- You can give database users selective `INSERT`-rights, so that the user cannot specify values for certain columns.

  So then you can be sure that date and user are stored as defined in the `DEFAULT`-clause.
- Of course, it is also possible to ensure that a column value must be explicitly defined: “`NOT NULL`” and no `DEFAULT`.

Default Values for Columns (3)

- In SQL-92 the default value can only be
  - a constant,
  - a function without arguments, or
  - `NULL`.
- In Oracle any term without column names is allowed.
- SQL-86 had no `DEFAULT`-clause.

Column Constraints (1)

- `NOT NULL`: The column must be defined (not a null value).
- `PRIMARY KEY`: This column is the primary key of the table.

  `PRIMARY KEY` implies `NOT NULL`, so you do not have to specify it. There can be only one primary key per table. In DB2 you must specify `NOT NULL` in addition.
- `UNIQUE`: This column is an alternative key for the table.

  `UNIQUE` does not imply `NOT NULL`.

  In DB2, `UNIQUE` can only be used together with `NOT NULL`.
- `REFERENCES ⟨Table⟩`: This column is a foreign key, values must appear in the primary key column of ⟨Table⟩.
- `CHECK (⟨Cond⟩)`: Values in this column must satisfy ⟨Cond⟩.

  SQL Server supports only one `CHECK`-constraint per column.

Column Constraints (2)

- ⟨Cond⟩ in `CHECK`-constraints looks like a `WHERE`-condition without subqueries, and without functions like `SYSDATE`.

  In column constraints, only this column can be accessed. Otherwise use a table constraint (see below).

  Oracle, SQL Server, and DB2 all exclude subqueries. The SQL-92 standard contains subqueries in `CHECK`-constraints, but then constraint checking is difficult/inefficient.
- The basic idea of efficient constraint checking is that the constraint was satisfied before the update.

  So here one only has to check the inserted/modified tuple. Since such constraints hold in the empty database state, and we ensure that updates do not destroy their validity, it follows by induction that they hold in every database state.

Column Constraints (3)

- You can optionally give a name to a constraint by prefixing it with “`CONSTRAINT ⟨Name⟩`”.
- Always define a name, otherwise the system chooses names like “`SYS_C036`”.
- When the constraint is violated, its name is printed, and system-generated names give unclear error messages.
- Dropping a constraint later is easier if you know its name.
- 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, you cannot give a name to `NOT NULL` constraints.

Column Constraints (4)

- Integrity constraints count as valid if they result in the truth value “unknown” (so attributes can be null).

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

- In SQL-86, only [`NOT NULL` [`UNIQUE`]] was supported.

  And `UNIQUE` was not implemented in many systems.

  In older code `CREATE UNIQUE INDEX` was often used to enforce key constraints. In 1989, the standard was extended by an optional “Integrity Enhancement Feature” (SQL-89).

Column Constraints (5)

```
CREATE TABLE Course(
   CRN     NUMERIC(5)
           CONSTRAINT Course_Key PRIMARY KEY
           CONSTRAINT CRN_Positive CHECK(CRN > 0),
   Title   VARCHAR(40)
           CONSTRAINT Title_defined NOT NULL,
   IName   VARCHAR(20)
           CONSTRAINT Inst_Ref REFERENCES Instructor,
   Day     CHAR(3)
           CONSTRAINT Valid_Day
              CHECK(Day='Mon' OR ...  OR Day='Sat'))
```


Column Constraints (6)

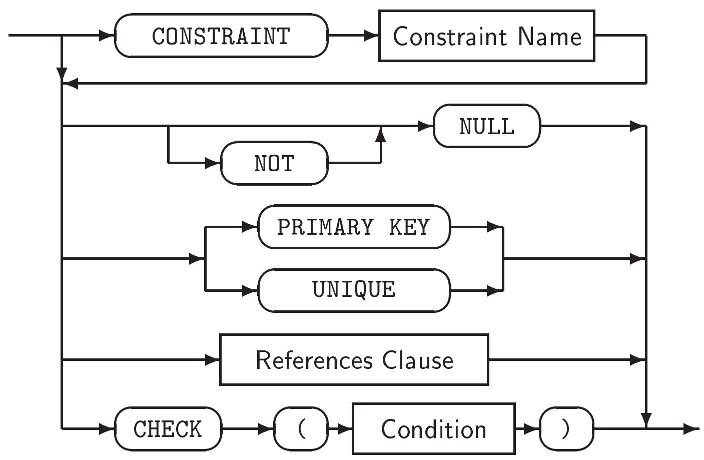


Table Constraints (1)

- 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.

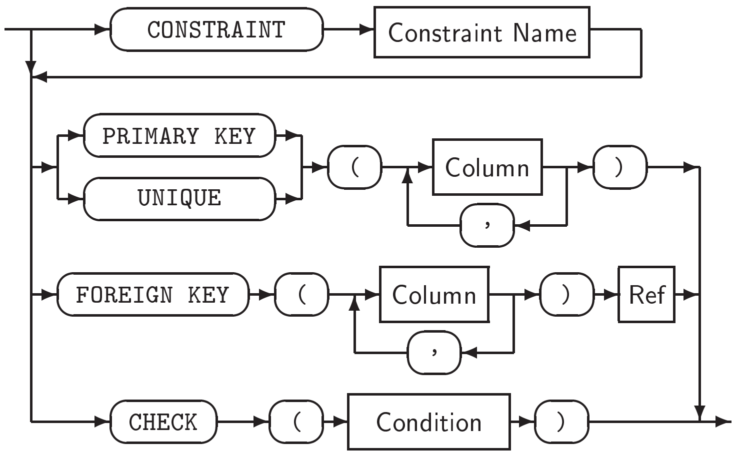
  Of course, you can define also constraints referring only to one column as table constraints, if you prefer that.

- `PRIMARY KEY(`A_1`, ..., `A_2`)`
- `UNIQUE(`A_1`, ..., `A_2`)`
- `FOREIGN KEY(`A_1`, ..., `A_2`) REFERENCES` R
- `CHECK(`φ`)`
- Syntactically, the difference is that table constraints are separated by a comma from column definitions and each other.

Table Constraints (2)

```
CREATE TABLE Course(
   CRN     NUMERIC(5)
           CONSTRAINT CRN_Positive CHECK(CRN > 0),
   Title   VARCHAR(40) NOT NULL,
   IName   VARCHAR(20) NULL,
   Day1    CHAR(3) CHECK(Day1 IN ('Mon', ..., 'Sat')),
   Day2    CHAR(3) CHECK(Day2 IN ('Mon', ..., 'Sat')),
   CONSTRAINT Course_Key PRIMARY KEY (CRN),
   CONSTRAINT Inst_Ref FOREIGN KEY (IName)
                       REFERENCES Instructor,
   CONSTRAINT Different_Days CHECK(Day1 <> Day2))
```


Table Constraints (3)



References Clause (1)

- The references clause is used after the column name in column constraints or after the `FOREIGN KEY` specification in table constraints, e.g. "`REFERENCES Instructor`".
- It is possible to specify the referenced column names, e.g. `REFERENCES Instructor(IName)`.

  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.

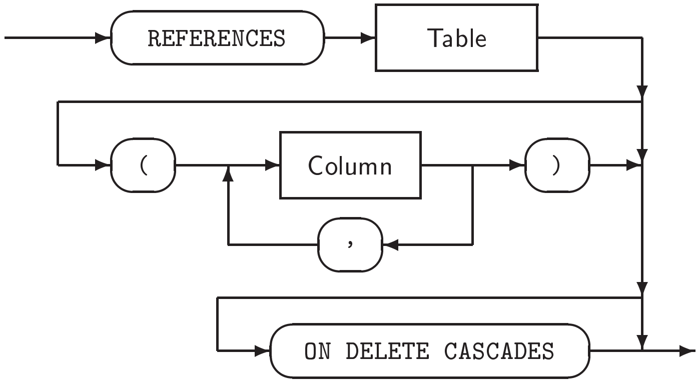
References Clause (2)

- One can request (in SQL-92, Oracle, DB2, not in SQL Server) that if an instructor is deleted, his/her courses are deleted, too:

  `REFERENCES Instructor ON DELETE CASCADES`
- Otherwise the system rejects attempts to delete an instructor who still has courses in the DB.
- In SQL-92 and DB2 (but not in Oracle or SQL Server), one can also request that if an instructor is deleted, his/her courses remain in the DB, but their attribute `IName` is set to a null value: "`ON DELETE SET NULL`".

  SQL-92 also allows "`SET DEFAULT`" and allows to specify the effects of updates on the key of instructor (name change): "`ON UPDATE ...`".

References Clause (3)



CREATE TABLE: Storage Parameters

- If you look up the syntax of the "`CREATE TABLE`" command in the SQL reference of your DBMS, you will find a long list of possible storage parameters, which can also 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 for you, you should try to understand the meaning of these parameters and set them.

  The defaults seldom give optimal performance.

CREATE TABLE: 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 and max. 900 bytes.

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 and max. 255 bytes.

CREATE TABLE: Restrictions (2)

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.

CREATE SCHEMA (1)

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

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

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

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

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

```
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, ...);
```


CREATE SCHEMA (2)

- 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 statement fails, all are rolled back (undone).
- Note that the single `CREATE TABLE` statements are not terminated with a ";" or in any other way.
- In DB2, schemas and users are separate things, although every schema belongs to exactly one user.

  `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.)

Deleting Tables (1)

- You can delete a table with the command

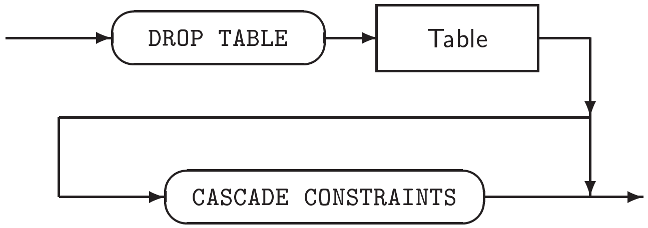
  `DROP TABLE ⟨Table Name⟩`

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

  Be careful! In Oracle, dropping a table automatically ends a transaction. So you cannot undo this action later.

- What if the table is referenced in foreign key constraints?
  - In SQL Server, you must first drop the referencing table.
  - In DB2, the foreign key is automatically dropped.
  - In Oracle, you specify "`CASCADE CONSTRAINTS`" if you want to drop the foreign keys, too.
  - In SQL-92, it is only "`CASCADE`".

Deleting Tables (2)



Examples:

- `DROP TABLE Instructor CASCADE CONSTRAINTS`
- `DROP TABLE Course`

  If the tables were dropped in the opposite order, no `CASCADE CONSTRAINTS` would be necessary.

Overview

1. Data Types
2. Table Definition
3. Update Commands
4. ALTER TABLE

Update Commands

- SQL has three commands to change the DB state:
  - INSERT: For inserting new rows in a table.
  - UPDATE: For changing attribute values of existing rows.
  - DELETE: For deleting rows from a table.
- In addition, SQL has two commands for ending transactions:
  - COMMIT: Transaction successful, make changes durable.
  - ROLLBACK: Transaction failed, undo all changes.

INSERT, 1st Form (1)

- The INSERT-command has two forms:
  - For inserting a single row with new data elements.
  - For inserting the result of a query into another table.
- First form:

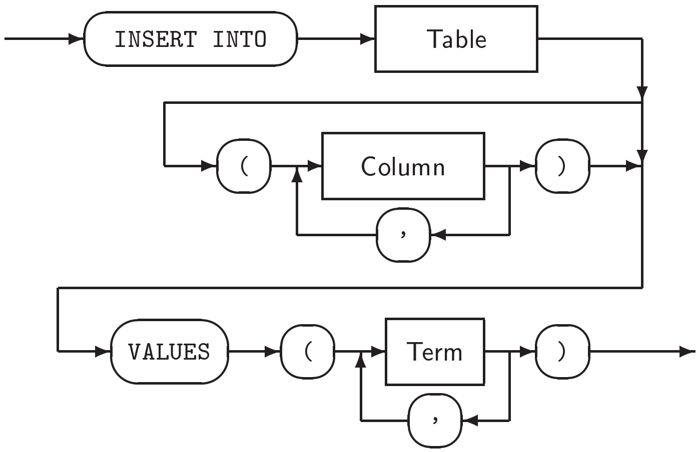
```
INSERT INTO Instructor
VALUES ('Brass, S.', 9404);
```

- Not only constants are allowed as values, but also “NULL”, “DEFAULT”, and any term like “2*3”, “SYSDATE”, etc.
- You can specify values for only a subset of the columns:

```
INSERT INTO Instructor(IName) VALUES ('Brass')
```

The default value is inserted in “Phone” (e.g. NULL).

INSERT, 1st Form (2)



INSERT, 2nd Form (1)

- Example:

```
INSERT INTO My_Courses(CRN, Title, Term)
SELECT CRN, Title, '99-2'
FROM   Course
WHERE  IName LIKE '%Brass%'
```

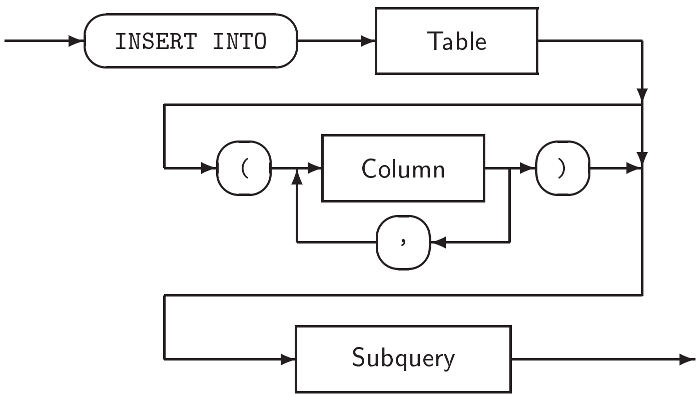
- The subquery will be evaluated before tuples are inserted.

  So you can use the table to be modified in the subquery with a defined result and without the risk of endless loops.

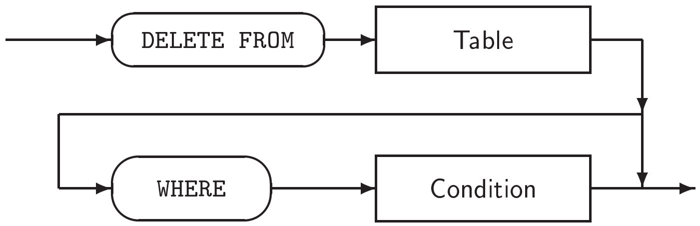
- So in this way you can copy tables or parts of tables.
- In SQL-92, there is only one general `INSERT` command.

  "`VALUES`" and "`SELECT`" are both table expressions.

INSERT, 2nd Form (2)



DELETE



- E.g.: Delete all courses by "`Brass`":

```
DELETE FROM Course
WHERE  IName LIKE '%Brass%'
```

- Be careful: Without the `WHERE`-condition, all tuples are deleted! (You might be able to use "`ROLLBACK`".)

TRUNCATE

- Oracle and SQL Server (not DB2) have a command

  `TRUNCATE TABLE ⟨Table Name⟩`

  which deletes all rows from the table and frees the disk space occupied by the table.

- In contrast to `DELETE`, it cannot be rolled back (undone). Therefore it is much faster.

  Also, at least in Oracle `DELETE` would not actually free any disk space. Another problem is that if you try to `DELETE` all rows of a large table, your rollback segment (containing the undo information) might be too small.

- `TRUNCATE` is not part of the SQL-92 standard.

  It does appear in the Oracle Certification exam.

UPDATE (1)

- The UPDATE command is for changing attribute values of selected tuples.
- E.g. give all employees from department 20 a 10% raise:

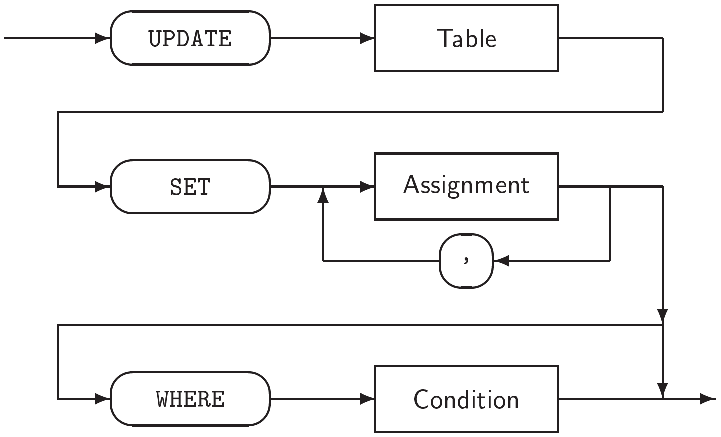
```
UPDATE Emp
SET    Sal = 1.1 * Sal
WHERE  Deptno = 20
```

- In SQL-92 and Oracle (but not in SQL-86), you can use a subquery to compute the new value.

  The subquery must return exactly one row (with a single column). If it returns no rows, a null value is used.

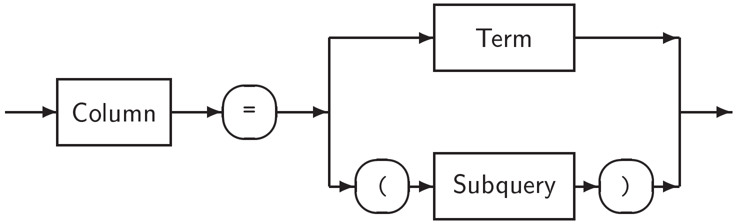
- You can change multiple columns in one UPDATE statement.

UPDATE (2)



UPDATE (3)

Assignment:



Transaction Management (1)

- SQL has no command for beginning a transaction.

  A transaction begins when you log into the database and every time after you finish a transaction.

- You finish a transaction with
  - COMMIT [WORK]: Makes changes durable.
  - ROLLBACK [WORK]: Undoes changes.

    Some commands, such as DROP TABLE, will automatically execute a COMMIT in Oracle, so they cannot be undone.

- If you want to try the “multi user” features of Oracle, you can start SQL*Plus several times from different windows.

  Uncommited data is not published, and locked for updates.

Transaction Management (2)

- If you are working with the database for a long time, you should COMMIT your work from time to time.

  If there should be a power failure etc., only the changes after the last COMMIT are lost. In addition, the DBMS might lock tuples for your transaction until you decide COMMIT or ROLLBACK. Very long transactions might hinder other users.
- In SQL*Plus you can "set autocommit on" which automatically does a commit after every update.

  However, then you cannot undo any changes.
- There is much more to say about transaction management and locking (see below). Also, at least Oracle does not reach the theoretical goal of "Serializability".

Overview

Changing Tables (1)

- Existing tables can be changed, e.g.
  - New columns can be added.

    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.

Changing Tables (2)

- SQL-86 had no ALTER TABLE, and every system varies slightly in what can be changed.
- If a table is small, it is usually easier to drop the table and recreate it.

  Of course, you must store the data in some temporary table (see "INSERT INTO SELECT") or export it to a file which you later load again.

  If a table is large, this might be a very costly operation.
- Also, if the table is referenced in foreign keys, views, indexes, triggers, stored procedures, grants, some of this information will be lost when the table is dropped.

ALTER TABLE in SQL-92 (1)

- You can add a column (which will be initially null):

```
ALTER TABLE Instructor
        ADD COLUMN Room NUMERIC(3)
```

- The keyword "COLUMN" is optional.
- If you specify a default value, the column can be NOT NULL:

```
ALTER TABLE Account ADD
Credit_Limit NUMERIC(7) DEFAULT 0 NOT NULL
```

- You can drop a column:

```
ALTER TABLE Instructor DROP COLUMN Room
```

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

ALTER TABLE in SQL-92 (2)

- Add a constraint (only table constraints can be added, but column constraints are anyway only syntactic shorthands):

```
ALTER TABLE Instructor
        ADD CONSTRAINT Room_defined
        CHECK(ROOM IS NOT NULL)
```

- You can remove a named constraint:

```
ALTER TABLE Instructor
        DROP CONSTRAINT Room_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 in Oracle (1)

- E.g. add a column "Room" to the table "Instructor":

```
ALTER TABLE Instructor ADD (Room NUMERIC(3))
```

- If you specify a default value, the column can be NOT NULL:

```
ALTER TABLE Account ADD
(Credit_Limit NUMERIC(7) DEFAULT 0 NOT NULL)
```

- Modify the datatype of a column:

```
ALTER TABLE Course MODIFY (Title VARCHAR(100))
```

- You cannot decrease the width of an existing column.
- You cannot use the MODIFY clause for changing column constraints except NULL/NOT NULL (but see below).
- You cannot drop or rename a column.

ALTER TABLE in Oracle (2)

- Add a constraint:

```
ALTER TABLE Course ADD
    (CONSTRAINT AltKey UNIQUE(Title))
```

  You must use the syntax for table constraints here.
- Drop a constraint (table or column constraint):

```
ALTER TABLE Course DROP CONSTRAINT AltKey
```

- If you do not know the constraint name, you can look it up in the data dictionary (USER_CONSTRAINTS).
- Drop a primary/alternative key:

```
ALTER TABLE Course DROP PRIMARY KEY CASCADE
ALTER TABLE Course DROP UNIQUE(Title)
```

ALTER TABLE in DB2 (1)

- Adding a column is done as in the SQL-92 standard:

```
ALTER TABLE Instructor
        ADD COLUMN Room NUMERIC(3)
```

  If you specify a default value, the column can be not null.

- There is no way to drop or rename a column.
- The only modification of a column is to change the size of a VARCHAR-column:

```
ALTER TABLE Course
    ALTER Title SET DATA TYPE VARCHAR(100)
```

  Again, only increases in size are possible.

ALTER TABLE in DB2 (2)

- 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.
- You can add a constraint as in the SQL-92 standard:

```
ALTER TABLE Instructor
        ADD CHECK(ROOM IS NOT NULL)
```

- You can drop a named constraint (basically as in SQL-92):

```
ALTER TABLE Course DROP CONSTRAINT Room_defined
```

- You can drop a primary key without name:

```
ALTER TABLE Course DROP PRIMARY KEY
```


ALTER TABLE in SQL Server (1)

- Adding a column is done basically as in the standard, however the keyword “COLUMN” after “ADD” is not allowed.

```
ALTER TABLE Instructor ADD Room NUMERIC(3)
```

  If you specify a default value, the column can be not null.

- You can drop a column (here “COLUMN” is required):

```
ALTER TABLE Instructor DROP COLUMN Room
```

- You can alter the data type of a column, but not if this column has a key or check constraint:

```
ALTER TABLE Course
    ALTER COLUMN Title VARCHAR(100)
```

  Decreases in size are possible if the data still fits.

ALTER TABLE in SQL Server (2)

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

```
ALTER TABLE Course
    ALTER COLUMN Title VARCHAR(100) NULL
```

- You can add a constraint as in the SQL-92 standard:

```
ALTER TABLE Instructor
        ADD CHECK(ROOM IS NOT NULL)
```

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

- You can drop a named constraint (basically as in SQL-92):

```
ALTER TABLE Course DROP CONSTRAINT Room_defined
```