

Part 5: Table Definition

References:

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Edition, 1999. Chap. 8, "SQL — The Relational Database Standard"
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Ed., McGraw-Hill, 1999. Chapter 4: "SQL".
- 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.
- 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.
- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data.

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`.

- explain why constraints are important.
- define not null, key, foreign key, and check constraints.

Overview

1. Data Types in SQL
2. Constraints: Overview
3. Key Constraints
4. Foreign Key Constraints
5. CREATE TABLE/ALTER TABLE Syntax

Example (1)

- Suppose the following table should be created:

COURSES				
CRN	TOPIC	TITLE	INSTRUCTOR	SEATS
22332	2710	DB Management	Flynn, R.	50
24271	2610	Data Structures	Flynn, R.	50
31864	2550	Client-Server	Spring, M.	30
41590	2711	DB ... Design	Brass, S.	70
37329	2711	DB ... Design	Brass, S.	30
25688	2770	Document Proc.		35

Example (2)

- This is done in SQL with the following statement:

```
CREATE TABLE COURSES(  
  CRN          NUMERIC(5)    NOT NULL  
                CHECK(CRN > 0),  
  TOPIC        NUMERIC(4)    NOT NULL  
                CHECK(TOPIC > 0),  
  TITLE        VARCHAR(40)   NOT NULL,  
  INSTRUCTOR   VARCHAR(20),  
  SEATS        NUMERIC(3)    CHECK(SEATS >= 0),  
  PRIMARY KEY (CRN),  
  FOREIGN KEY (INSTRUCTOR) REFERENCES FACULTY)
```

Example (3)

- The `CREATE TABLE`-statement defines the name of the table, the columns with their data types, and constraints.
- Constraints are conditions that the data must satisfy, e.g. `CRN > 0`. The DBMS will not accept data that violate constraints.
- After the table is created, it can be filled with statements like the following:

```
INSERT INTO COURSES VALUES  
(22332, 2710, 'DB Management', 'Flynn, R.', 50)
```

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 allow user-defined data types.

DB2, Oracle, and SQL Server have some support for 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 does not work in SQL Server 7 (one must write "+" there).

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.

Character Strings (2)

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

Of course, Oracle, SQL Server, and DB2 support it.

- The systems differ in the maximal possible value for the string length *n*:

DBMS	Maximal <i>n</i>
Oracle 8.0	2000
DB2 UDB 5	254
SQL Server 7	8000

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.
- `VARCHAR` is supported in Oracle, SQL Server, DB2.

Probably all modern DBMS support it.

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 <i>n</i>
Oracle 8.0	4000
DB2 UDB 5	254/4000
SQL Server 7	8000

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

String Comparisons (1)

- The outcome of comparing ($=$, $<>$, $<$, $<=$, $>$, $>=$) two character strings may depend on the DBMS.

Or settings within the DBMS.

- The SQL-92 standard defines the notion of “collation sequences” (or “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.

String Comparisons (2)

- 'a' < 'b' etc., and 'A' < 'B' etc. can be expected. But already comparisons between uppercase and lowercase characters are not so clear.
- The default 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'.
 - ◇ SQL Server is case-insensitive, e.g.: 'a' = 'A'.

String Comparisons (3)

- It might be possible to change this, but e.g. only during installation (SQL Server), or during database creation (Oracle, DB2).
- 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.

Actually, DB2 makes two passes: It first compares the character “weights”, and if there is no difference, also the character codes.

String Comparisons (4)

- 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.

String Comparisons (5)

- DB2 and SQL Server seem to use always the blank-padded semantics (at least by default).
- Oracle uses the nonpadded semantics if at least one operand of a comparison 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. a comparison of `CHAR(10)` and `CHAR(20)` columns can possibly yield “true” as can a comparison of 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. Trailing spaces in `VARCHAR2`-columns can be quite annoying: They are not visible in the output, but the comparison does not work.

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(s1 IN s2)`.

- `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:

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

String Functions (4)

String functions in DB2, Continued:

- `ASCII(s)`, `CHR(n)`.
- `DIFFERENCE(s1, s2)`, `SOUNDEX(s)`.
- `GENERATE_UNIQUE()`, `REPEAT(s, n)`, `SPACE(n)`.

Numbers (1)

- $\text{NUMERIC}(p, s)$: 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. $\text{NUMERIC}(3, 1)$ allow one to store -99.9 to 99.9 .
- $\text{NUMERIC}(p)$: Signed whole number of p digits.

Whole numbers are also known as “integers”. I.e. $\text{NUMERIC}(p)$ is an abbreviation for $\text{NUMERIC}(p, 0)$.

“NUMERIC” without p uses an implementation-defined p .

- E.g. $\text{NUMERIC}(2)$ allows one to store -99 to 99 .

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).

`DECIMAL` can be abbreviated “`DEC`”.

- These data types were already contained in the SQL-86 standard and are supported in all DBMS.
- Oracle normally uses `NUMBER(p, s)` and `NUMBER(p)`, but understands `NUMERIC/DECIMAL` as synonyms.

Neither DB2 nor SQL Server understand `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

In SQL Server, the server must be started with the option `/p` to support up to 38 digits. Otherwise the limit is 28 digits.

- 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 and SQL Server use 32 bit binary numbers: $-2147483648(-2^{31})$.. $+2147483647(2^{31}-1)$. In Oracle, it is a synonym for `NUMBER(38)`. `INTEGER` can be abbreviated "INT".

- **SMALLINT**: As above, range of values may be smaller.

DB2 and SQL Server use 16 bit binary numbers: $-32768(-2^{15})$.. $+32767(2^{15}-1)$. In Oracle, it is again 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 (5)

- $\text{FLOAT}(p)$: 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 $\text{FLOAT}(p)$ with implementation defined values for p .
- E.g. SQL Server (DB2 is similar):
 - ◇ $\text{FLOAT}(p)$, $1 \leq p \leq 24$, uses 4 Bytes.
7 digits precision (range $-3.40\text{E}+38$ to $3.40\text{E}+38$).
 REAL means $\text{FLOAT}(24)$.
 - ◇ $\text{FLOAT}(p)$, $25 \leq p \leq 53$, uses 8 Bytes.
15 digits precision (range $-1.79\text{E}+308$ to $+1.79\text{E}+308$).
 DOUBLE PRECISION means $\text{FLOAT}(53)$.

Numbers (6)

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

Oracle also understands `FLOAT(p)`. `NUMBER` allows storage of values between $1.0 * 10^{-130}$ and $9.9 \dots * 10^{125}$ 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, never use `FLOAT`.

Numbers (7)

Operations for Numbers in Oracle:

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

Numbers (8)

Operations for Numbers in the SQL-92 Standard:

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

Operations for Numbers in DB2:

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

Numbers (9)

Operations for Numbers in DB2, continued:

- $\text{EXP}(x)$, $\text{LN}(x)$, $\text{LOG10}(x)$, $\text{POWER}(x, y)$.
- $\text{MOD}(m, n)$.
- $\text{SQRT}(x)$.
- $\text{RAND}()$.
- $\text{DEGREES}(x)$.

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.
They also understand VARCHAR, which was not contained in SQL-86.

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. 2147483647.

- 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`.

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 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.

Normally `CHAR(1)` is used together with a constraint that only 'Y' and 'N' are allowed in this column.

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.).

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 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).

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.

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).

Overview

1. Data Types in SQL

2. Constraints: Overview

3. Key Constraints

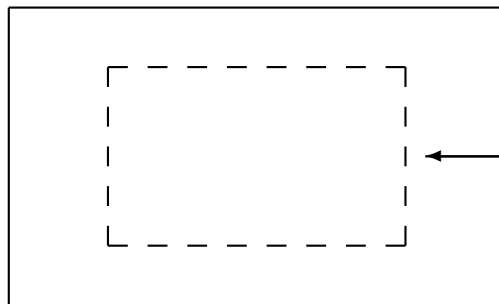
4. Foreign Key Constraints

5. CREATE TABLE/ALTER TABLE Syntax

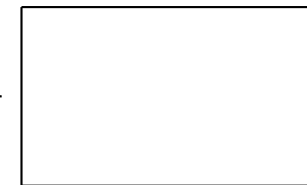
Valid Database States (1)

- Goal of DB-Design: DB-States should be image of real world.
- But the definition of tables and columns allows many database states which do not correspond to states of the real world:

DB-States for Schema



Possible Situations
in the real world



← isomorphic →

Valid Database States (2)

CUSTOMER					
Cust_No	Name	Birth_Year	City	...	
1	Smith	1936	Pittsburgh	...	
2	Jones	1965	Philadelphia	...	
3	Brown	64	New York	...	
3	Ford	1990	Washington	...	

- Customer numbers should be unique.
- The year of birth must be greater than 1870.
- Customers must be at least 18 years old.

Valid Database States (3)

- It might be that the DB state corresponds to a different situation of the real world than the actual one.

In this case only the DB state is wrong (not the schema).

- Protection against such errors is difficult.

Two independent persons must enter the same data, the supervisor has to sign it off, etc.

Valid Database States (4)

- Expect protection against entering data which does not make sense or is illegal (impossible in the real world).
- Such data would make DB state inconsistent with the general understanding of the real world, which could result in strange effects.

If this is possible, the DB schema is wrong.

- Constraints are used to avoid this type of errors.

Constraints (1)

- Integrity Constraints (or short “Constraints”) are conditions which every database state must satisfy.
- This restricts the set of possible database states.

Ideally to images of possible real world situations.

- The DBMS will refuse any change which would violate such a constraint.

In this way, invalid states are excluded.

Constraints (2)

- “Business rules” are similar to constraints: They are rules that must be followed by employees (to prevent chaos).
- Constraints are used to implement “business rules” .

Certain rules, e.g. that customers must be 18 years old or that advanced cryptology software is not sold to customers outside the US, can be enforced by not allowing entry of such an order into the database. A state violating the business rules is considered invalid.
- However, there are also “business rules” that are implemented via the basic table structure, access rights, application programs, etc.

Constraints (3)

- In the SQL `CREATE TABLE` statement, certain common types of constraints can be specified:
 - ◇ `NOT NULL` constraints: A column cannot be null.
 - ◇ Key constraints: Unique identification of rows.
 - ◇ Foreign key constraints: Values in a column must also appear as a key in another table.
 - ◇ `CHECK` constraints: Column values or rows must satisfy a condition.

The condition can only refer columns of a single row at a time.

Constraints (4)

- The SQL-92 standard contains a general `CREATE ASSERTION` statement, which is, however, not implemented in today's database systems.
- One can formalize general constraints e.g. as SQL queries that return the violations or one can at least document them in natural language.

Of course, the DBMS cannot understand and thus cannot enforce constraints in natural language. However, they can still be an important documentation for the later development of application programs. If constraints are formulated as SQL queries that return violations, one can run them from time to time.

Constraints (5)

Exercise: Which of the following are constraints?

- It is possible that an instructor teaches no course.

Yes No

- For every course, the number of registered students must be less than (or equal to) the number of seats in the room in which the course will be held (assuming that course registration and room information is contained in the DB).

Yes No

Constraints (6)

Exercise, continued:

- No teacher can teach two courses at the same time (assuming that the course schedule is contained in the DB).

Yes No

- The numeric codes for the attribute “Type” of “Instructor” mean the following: 1: Teaching Fellow, 2: Adjunct Faculty, 3: Assistant Professor, 4: Associate Professor, 5: Full Professor.

Yes No

Constraints (7)

Exercise, continued:

- Course titles should be unique.
 Yes No
- For every instructor, the year in which he/she got the PhD must be greater than his/her birth year.
 Yes No
- An instructor teaches either no course, or one course, or more than one course.
 Yes No

Trivial/Implied Constraints

- A trivial constraint is a condition which is always satisfied (logically equivalent to “true”).
- A constraint A logically implies a constraint B if whenever A is true, also B is true.

I.e. the states satisfying A are a subset of the states satisfying B .

- E.g. A : “Every instructor teaches 1 or 2 courses” .
 B : “Instructors can teach at most 4 courses” .
- Trivial and implied constraints should not be specified.

Reasons for Constraints (1)

- Constraints provide some protection against data input errors (obviously wrong values are excluded).
- Constraints document knowledge about the DB.
- Constraints help to enforce company standards.
- If the schema should contain redundant information, constraints ensure that the copies agree.
- Queries and programs are simpler if the programmer is not required to handle exceptional cases.

E.g. a program for printing course information becomes simpler if we know that each course has only one instructor.

Reasons for Constraints (2)

- If one assumes that the data fulfills some condition, but it actually does not, strange effects can occur:
 - ◇ E.g. the programmer assumes that a certain column cannot contain null values.
 - ◇ So he/she uses no indicator variable.
 - ◇ As long as there are no null values, this works.
 - ◇ But if no constraint prevents this, after some time somebody will enter a null value.
 - ◇ Then the program will break with a not user-friendly Oracle error message.

Constraints vs. Exceptions (1)

- Constraints cannot have exceptions.
- A good DBMS will reject any attempt to enter data which violates a specified constraint.
- It is a common experience that there will be exceptional situations in which the DB system seems inflexible because of the specified constraints.
- The introduction of a computerized system always changes the real world.

So the decreased flexibility is not necessarily fatal, if the users can accept it.

Constraints vs. Exceptions (2)

- Without constraints, the database contents will become chaotic. However, only unquestionable conditions should be defined as constraints.
- “Mostly true” conditions should be documented, but not declared as a constraint.
 - ◇ Such conditions might be interesting in physical database design.
 - ◇ Application programs might ask the user for an additional confirmation when unlikely data is entered, e.g. a new customer older than 100 years.

Summary

- The three basic design errors are:
 - ◇ Situations in the real world which do not correspond to a database state.

I.e. data which actually occurs cannot be entered.
 - ◇ A legal question about the real world cannot be formulated as a query to the database.

I.e. information is missing in the database.
 - ◇ Database states are possible which do not correspond to a legal state in the real world.

I.e. invalid/illegal data can be entered.

“Not Null” Constraints (1)

- Since null values lead to complications, it can be specified for each attribute whether or not a null value is allowed.
- It is important to invest careful thought as to where null values are needed.
- Declaring many attributes “not null” will result in simpler programs and fewer surprises with queries.
- However, flexibility is lost: Users are forced to enter values for all “not null” attributes.

“Not Null” Constraints (2)

- The relational model allows null values for all columns, but constraints that null values are invalid in certain columns can be specified.
- In SQL, one writes `NOT NULL` after the data type for an attribute which cannot be null.

This is a kind of “column constraint” (see below). Between the data type and the “`NOT NULL`” one could write other column constraints (e.g. `CHECK`), but usually “`NOT NULL`” is specified first. In this way, it can also be seen as part of the data type.

- Otherwise, attributes do accept null values.

“Not Null” Constraints (3)

- When discussing relational DB schemas, the entire `CREATE TABLE` statement may be a bit long.
- One can e.g. use the following simplified notation that shows only the table name and the column names, but not the column data types:

Instructor(Name, Room, Phone, HomePhone^o)

- In this notation, attributes which can take a null value must be explicitly marked with a small “o” (optional) in the exponent.

“Not Null” Constraints (4)

- For tabular notation, the possibility of null values can be indicated in one of these ways:

Instructor		
Column	Type	Null
Name	char(30)	n
⋮	⋮	⋮
HomePhone	char(13)	y

Instructor	Name	Room	OfficePhone	HomePhone
Type	char(30)	char(9)	char(5)	char(13)
Null	n	n	n	y

Overview

1. Data Types in SQL
2. Constraints: Overview
3. Key Constraints
4. Foreign Key Constraints
5. CREATE TABLE/ALTER TABLE Syntax

Keys (1)

- A key of a relation R is an attribute/column A that uniquely identifies the tuples/rows in R .

I.e. there may never be two different tuples t and u in R such that $t.A = u.A$.

- E.g. if SSN has been declared as key of students, this database state is illegal:

STUDENTS		
<u>SSN</u>	FIRST	LAST
111-22-3333	John	Smith
111-22-3333	Ann	Miller
900-50-3000	Mary	Jones

Keys (2)

- If SSN has been declared as key of STUDENTS, the DBMS will refuse the insertion of a second row with the same value for SSN as an existing row.
- Note that keys are constraints: They refer to all possible DB states, not only the current one.
- Even though in the above database state (with only three students) the last name (LAST) could serve as a key, this would be too restrictive: E.g. the future insertion of “John Miller” would be impossible.

Keys (3)

- A key can also consist of several attributes. Such a key is called a “composite key”.

If A and B together form a key, it is forbidden that there are two rows t and u which agree in both attributes (i.e. $t.A = u.A$ and $t.B = u.B$). They may agree in one attribute as long as they differ in the other.

- E.g. this relation satisfies the key FIRST, LAST :

STUDENTS		
<u>SSN</u>	<u>FIRST</u>	<u>LAST</u>
123-45-6789	John	Smith
111-22-3333	Ann	Miller
125-33-1248	John	Miller

Keys (4)

Implication Between Key Constraints:

- Key constraints become weaker (i.e. less restrictive, more DB states are valid) if attributes are added.
- E.g. the relation on the previous slide:
 - ◇ violates the key constraint **FIRST**,
There are two “Johns” .
 - ◇ violates the key constraint **LAST**,
There are two “Millers” .
 - ◇ satisfies the composite key constraint **FIRST, LAST**.

Keys (5)

Minimality of Keys:

- If one has declared a key, e.g. SSN, then any superset of it (e.g. SSN together with LAST) automatically satisfies the “unique identification” property.

If there cannot be two students with the same SSN, there certainly cannot be two students that have the same SSN and the same last name.

- The usual definition of a key requires that the set of key attributes $\{A_1, \dots, A_k\}$ is minimal.

Since the unique identification property automatically holds for supersets, “nonminimal keys” are indeed not interesting.

Keys (6)

Minimality of Keys, Continued:

- The minimality requirement means that we cannot leave out any of the key attributes without destroying the property of unique identification.
- However, in this definition, a key is not only
 - ◇ a constraint, which excludes invalid DB states,
 - ◇ but also an assertion about the real world, that certain states are possible.
- In the literature, a set of attributes that only satisfies the unique identification is called a “superkey” .

Keys (7)

Multiple Keys:

- A relation may have more than one key.
- E.g. SSN is a key of STUDENTS.
- But if the table contains only students in a single course or a very small school, it might make sense to declare also FIRST and LAST together as key.
- Both keys are minimal, because neither is a subset of the other. (None of the two implies the other.)

It does not matter that the first key consists of only one attribute, whereas the second consists of two.

Keys (8)

Multiple Keys, Continued:

- One of the keys is designated as the “primary key” .
- Other keys are called “alternate/secondary keys” .

SQL uses the keyword `UNIQUE` for alternate keys.

- The primary key should be a key that consists only of a single, short attribute and is never updated (if available).

The primary key is used in other tables that need to refer to rows in this table. In some systems, access via the primary key might be especially fast. Otherwise, the selection of the primary key is more or less arbitrary.

Keys (9)

- The primary key attributes are often marked by underlining them in relational schema specifications:

$$R(\underline{A_1: D_1}, \dots, \underline{A_k: D_k}, A_{k+1: D_{k+1}}, \dots, A_n: D_n).$$

STUDENTS		
<u>SSN</u>	FIRST	LAST
123-45-6789	John	Smith
111-22-3333	Ann	Miller
900-50-3000	Mary	Jones

- Usually, the attributes of a relation are ordered such that the primary key consists of the first attributes.

Keys (10)

Keys and Null Values:

- The primary key cannot be null, other keys should not be null.

In SQL-89 and DB2, `NOT NULL` must be specified for every attribute in a `PRIMARY KEY` or `UNIQUE` constraint.

In SQL-92 and Oracle, the “`PRIMARY KEY`” declaration automatically implies “`NOT NULL`”, but “`UNIQUE`” (for alternate keys) does not.

In Oracle, there can be several rows, all with a null value in a `UNIQUE` key. In SQL Server, only one row can be null. SQL-92 defines three different semantics for composite keys which have null values in only some of their attributes. One should avoid all this mess.

- It is as not acceptable if already the “object identity” of the tuple is not known.

Keys (11)

Keys and Updates:

- It is considered poor style if key attribute values are modified (updated).

This would change the “object identity”. Better: Delete the tuple first and then insert a tuple with the new values.

- But SQL does not enforce this constraint.

The standard even contains specifications for what to do with foreign keys if the referenced key value is updated.

Keys (12)

The Weakest Possible Key:

- A key consisting of all attributes of a relation requires only that there can exist no two different tuples which agree in all attributes.

Theoretically, relations are sets: Then this is no restriction. However, in practice, relations are multisets (bags), and this key ensures that there are no duplicates.

- **Style Recommendation:** Define at least one key for every relation in order to exclude duplicate tuples.

If there is no other key, define the key consisting of all attributes of the relation.

Keys (13)

Summary:

- Declaring a set of attributes as a key is a bit more restrictive than the unique identification property:
 - ◇ Null values are excluded at least in the primary key.
 - ◇ One should avoid updates, at least of the primary key.
- However, the uniqueness is the main requirement for a key. Everything else is secondary.

Exercises (1)

- Select a key:

HOMEWORK_RESULTS		
Student	Exercise	Points
John Smith	1	10
John Smith	2	12
Maria Brown	1	9

- Give an example for an insertion that would violate the key:

--	--	--

- Could “Points” also be declared as key?

Exercises (2)

- Consider an appointment calendar:

APPOINTMENTS				
Date	From	Until	Room	What
Jan. 19	10:00	11:00	IS 726	Michael
Jan. 19	14:00	15:00	IS 726	Siripun
Jan. 19	18:00	20:50	IS 501	INFSCI 2710

- What would be correct keys?
- Give an example for a non-minimal key (superkey).
- Are additional constraints needed? I.e. can there be invalid database states, even if the key is satisfied?

Exercises (3)

- Suppose a table for the faculty members of a school (or department) has to be designed.
- Somebody proposed to choose the combination of `FIRST_NAME` and `LAST_NAME` as a key.
- Somebody else says that this is not possible, since there can be sometime in the future two professors with the same name.
- What do you think about this?

Would the situation be any different if the table should contain all students of the university?

More About Keys (1)

- If there is no natural key, identifying numbers can be added.

E.g. “order number”, “course number”.

- The selection of a natural key (not an artificial identification number) is quite difficult.

One of Murphy's Laws: There are always exceptions.

- Often, thinking about a key helps to understand the real meaning of a concept better.

E.g. course offering vs. abstract concept of a course (topic).

More About Keys (2)

- Even if an artificial number is used, think about the identification mechanisms used in the application programs.

It is dangerous if different application programs use different identification mechanisms for the same thing.

- If the non-presence in the database is used to make statements about other objects in the real world (e.g. “bad credit” DB), all these objects must be uniquely identified.

The objects about which information is stored might violate the key although the subset in the database satisfies it.

More About Keys (3)

- The purpose of keys is not only the identification of entities.
- Keys should also help to avoid duplicates in the database.
- Often, duplicates are not exact copies: For instance, other shorthands or capitalization is used.

Of course, the pure concept of a key then does not help. Furthermore, a constraint is not really needed — some warning would suffice.

- Think about possibilities for detecting duplicates before writing application programs.

Overview

1. Data Types in SQL
2. Constraints: Overview
3. Key Constraints
4. Foreign Key Constraints
5. CREATE TABLE/ALTER TABLE Syntax

Foreign Keys (1)

- The relational model has no explicit relationships, links or pointers.
- Values for the key attributes identify a tuple.
They are “logical addresses” of the tuples.
- To refer to tuples of R in a relation S , include the primary key of R among the attributes of S .
Such attribute values are “logical pointers” to tuples in R .
- E.g. the ENROLLMENTS table contains the attribute CRN, which is the primary key of COURSES.

Foreign Keys (2)

CRN in ENROLLMENTS is a foreign key referencing COURSES:

ENROLLMENTS			COURSES		
<u>SSN</u>	<u>CRN</u>		<u>CRN</u>	TOPIC	TITLE
543-76-9821	22332	→	22332	2710	DB Management
111-22-3333	24271	→	24271	2610	Data Structures
543-76-9821	24271	→	31864	2550	Client-Server
543-76-9821	31864	→	37329	2711	DB ... Design
123-45-6789	41590	→	41590	2711	DB ... Design
111-22-3333	41590	→			
111-22-3333	50000	→?			

Error

The constraint that is needed here is that every CRN value in ENROLLMENTS also appears in courses.

Foreign Keys (3)

- Declaring CRN in ENROLLMENTS as a foreign key that references COURSES means that the DBMS will reject any attempt to insert an enrollment for a non-existing course.
- So the set of CRN-values that appear in COURSES are a kind of “dynamic domain” for the attribute CRN in ENROLLMENTS.

The set of values that appear in CRN in ENROLLMENTS must be a subset of the set of values that appear in CRN in COURSES.

Foreign Keys (4)

- Tuples in `ENROLLMENTS` are combined with tuples in `COURSES` by means of a join.

This application of the join corresponds to the dereferencing of pointers in other models.

- The foreign key constraint ensures that every row in `ENROLLMENTS` really has a join partner in `COURSES`.

Without foreign key constraint, there could be “dangling pointers” that point to nowhere.

- The key constraint for `COURSES` ensures that the join partner is unique.

Foreign Keys (5)

- One notation for relational schemas denotes a foreign key by an arrow and referenced relation:

COURSES(CRN, TOPIC, TITLE)

STUDENTS(SSN, FIRST, LAST)

ENROLLMENTS(SSN → STUDENTS, CRN → COURSES)

- A foreign key corresponds to a “one-to-many” relationship: One course is taken by many students.
- However, in this example, two “one-to-many” relationships are composed to a “many-to-many” relationship between students and courses (see below).

Foreign Keys (6)

- The table `ENROLLMENTS` which contains the foreign key is also called the “child” table of the referential integrity constraint, and the referenced table `COURSES` is the “parent table”.
- It is not required that the foreign key attribute has the same name as the referenced primary key attribute: Unless one explicitly specifies the names of the referenced attribute, the primary key is meant.

It is possible in SQL to reference alternate keys, but that is probably bad style. It is not possible to reference non-key attributes.

Foreign Keys (7)

- A foreign key can consist of multiple attributes, e.g. (FIRST, LAST).

Then the value combinations for the two columns must also appear in the referenced table.

- E.g. the table COURSES may reference the instructor by first and last name:

```
COURSES(CRN, TOPIC, TITLE,  
        (INST_FIRST, INST_LAST) → FACULTY)  
FACULTY(FIRST, LAST, OFFICE, PHONE, EMAIL)
```

- The parentheses are required in this notation.

Foreign Keys (8)

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

Columns are matched by their position in the declaration: E.g. if the key is (FIRST, LAST) and the foreign key is (LAST, FIRST) insertions will very probably give an error. If the data types of FIRST and LAST are sufficiently different, the error might be detected when the foreign key is declared. But some systems require only “compatible” data types.

- Only keys can be referenced: One cannot reference only part of a composite key or a non-key attribute.

Normally, one should reference only the primary key, but SQL permits referencing alternate keys.

Foreign Keys (9)

Foreign Keys and Null Values:

- Unless a “not null” constraint is explicitly specified, foreign keys can be null.
- The foreign key constraint is satisfied even if the referencing attributes are “null”. This corresponds to “nil” pointer.
- If a foreign key consists of more than one attribute, they should either all be null, or none should be null.

But Oracle and SQL-92 allow partially defined foreign keys. In Oracle, if at least one attribute in the foreign key is null, the constraint counts as satisfied. The SQL-92 standard defines three different semantics.

Foreign Keys (10)

- It is possible that parent and child are the same table, e.g.

EMP(EMPNO, ENAME, JOB, MGR^o→EMP, DEPTNO→DEPT)

PERSON(NAME, MOTHER^o→PERSON, FATHER^o→PERSON)

- In this notation, attributes that can be null (“optional attributes”) are marked with “^o”.
- Two relations can reference each other, e.g.
EMPLOYEE(EmpNo, ..., Dept→DEPT)
DEPT(DNo, ..., Leader^o→EMPLOYEE).
- Exercise/Puzzle: How can tuples be inserted?

Please Remember:

- Foreign keys are not themselves keys!

The attributes which form a foreign key may be part of a key, but this is an exception, not the rule. The foreign key constraint has nothing to do with a key constraint.

For some authors, however, a key is any attribute that identifies tuples, not necessary of the same relation. Then foreign keys would be keys, but normal keys need some adjective.

- Only a key of a relation can be referenced, not arbitrary attributes.
- If the key of the referenced relation consists of two attributes, the foreign key must also consist of two attributes of the same data types in the same order.

Foreign Keys and Updates (1)

The following operations can violate a foreign key:

- Insertion into the child table `ENROLLMENTS` without a matching tuple in the parent table `COURSES`.
- Deletion from the parent table `COURSES` where the deleted tuple is still referenced.
- Modification of the foreign key `CRN` in `ENROLLMENTS` to a value not in `COURSES`.

This is usually treated like an insertion.

- Modification of the key of the parent table `COURSES` where the old value is still referenced.

Foreign Keys and Updates (2)

Note:

- Deletions from ENROLLMENTS (child) and insertions in COURSES (parent) can never lead to a violation of the foreign key constraint.

This means that the DBMS does not have to check the constraint for these operations.

Reactions on Insertions of Dangling References:

- The insertion is rejected. The DB state remains unchanged.

Foreign Keys and Updates (3)

Reactions on Deletions of Referenced Key Values:

- The deletion is rejected.
- The deletion cascades: All tuples from `ENROLLMENTS` which reference the deleted `COURSES` tuple are deleted, too.
- The foreign key is set to null.

Contained in SQL-92, supported in DB2, not in Oracle.

- The foreign key is set to a declared default value.

Contained in SQL-92, but not in Oracle or DB2.

Foreign Keys and Updates (4)

Reactions on Updates of Referenced Key Values:

- The update is rejected. The DB state remains unchanged.

DB2 and Oracle support only this alternative of the SQL-92 standard. In any case, changing key attributes is bad style.

- The update cascades.

I.e. the composer number is changed in the Piece table in the same way as it was changed in the Composer table.

- The foreign key is set to null.
- The foreign key is set to a declared default value.

Foreign Keys and Updates (5)

- When specifying a foreign key, decide which reaction is best.
- The default is the first alternative (“No Action”).
- At least for deletions from the parent table, all systems should support also the cascading deletion.

This is a kind of active integrity enforcement: The system does not reject the update, but does some other updates in order to repair the database state.

- Other alternatives exist only in few systems at the moment.

Exercise (1)

Define a relational DB schema for hotel. It requires:

- Information about guests: First Name, Last Name, and Home Address.
- Information about rooms: Is it single or double? What is the official room rate? When was it last renovated?
- Information about stays: Which room was rented by which guest from which date to which date? And at what room rate was he/she charged?

It might be less than the official rate.

Exercise (2)

Please define the following:

- Table names and column names.
- Keys.
- Foreign keys.
- Null constraints.

Furthermore, describe any additional constraints which might be necessary.

Another Schema Notation

- In the Oracle DBA Exam, the following “Instance Chart” is used to describe a relation schema:

Instance Chart for Table PIECE			
Column Name:	PNO	PNAME	CNO
Key Type:	PK		FK
Nulls/Unique:	NN, U	NN	
FK Table:			COMPOSER
FK Column:			CNO
Datatype:	NUMBER	VARCHAR	NUMBER
Length:	4	40	2

- “FK” stands for “foreign key”, “NN” for “not null”, “PK” for “primary key”, “U” for “unique”.

Overview

1. Data Types in SQL
2. Constraints: Overview
3. Key Constraints
4. Foreign Key Constraints

5. CREATE TABLE/ALTER TABLE Syntax

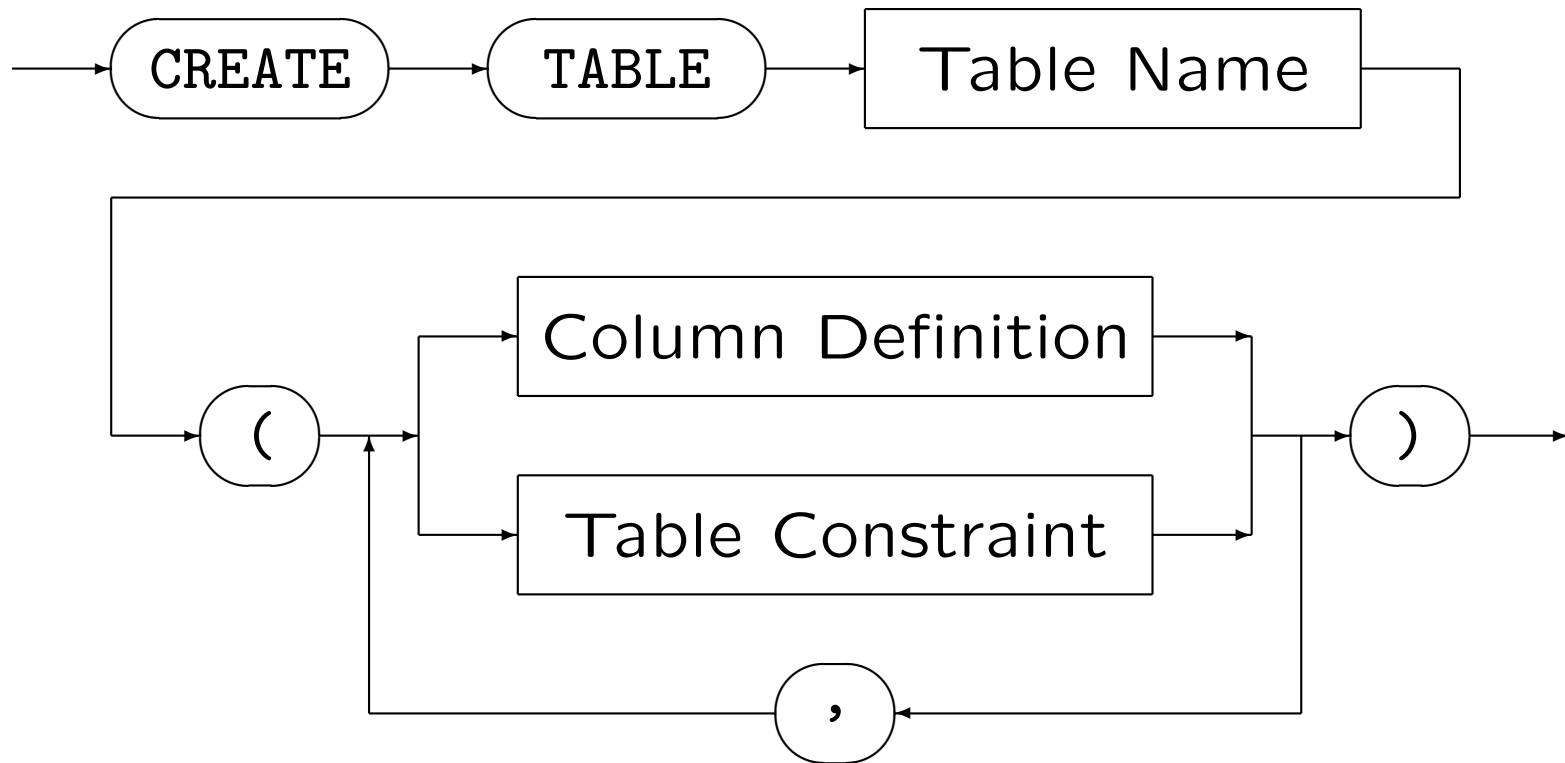
Example

- ENROLLMENTS (SSN→STUDENTS, CRN→COURSES):

ENROLLMENTS	
<u>SSN</u>	<u>CRN</u>
123-45-6789	41590
111-22-3333	41590
111-22-3333	31864

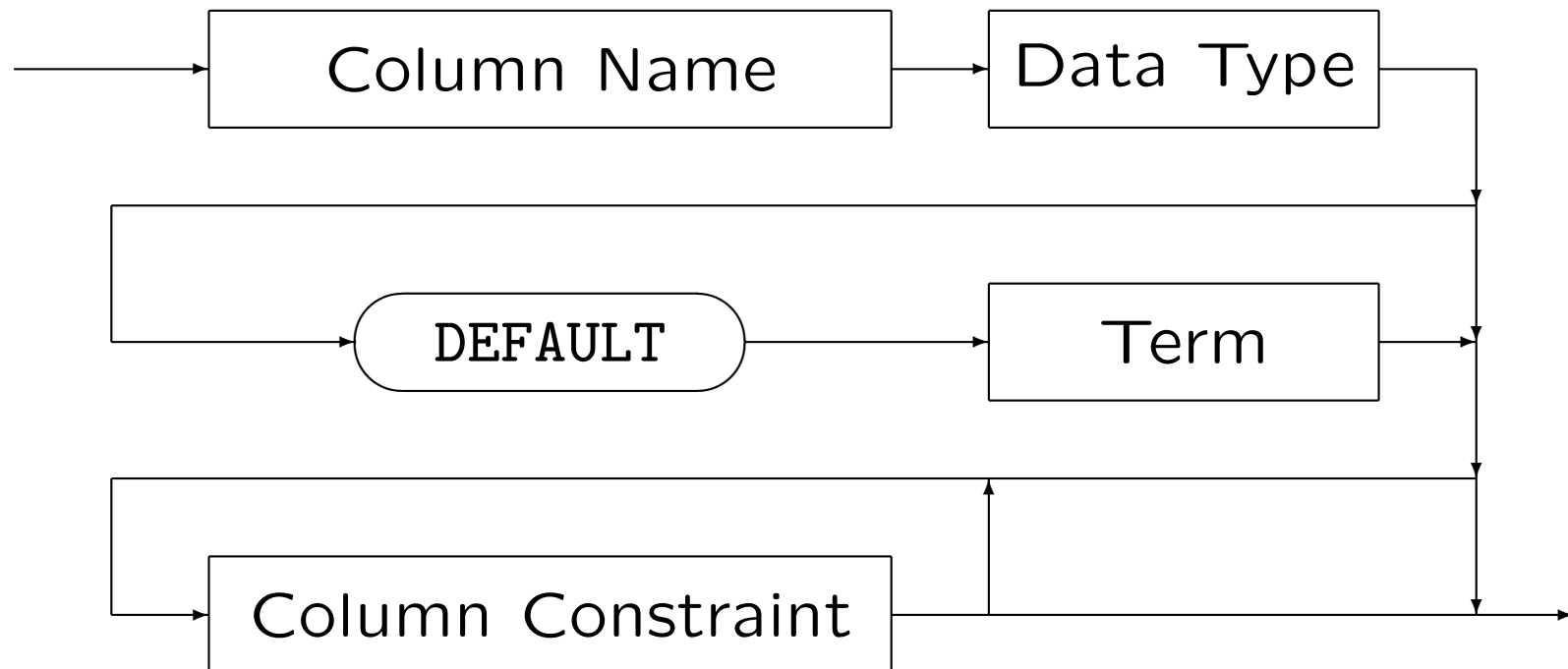
- CREATE TABLE ENROLLMENTS(
SSN CHAR(11) NOT NULL REFERENCES STUDENTS,
CRN NUMERIC(5) NOT NULL REFERENCES COURSES,
PRIMARY KEY (SSN, CRN))

CREATE TABLE: Syntax (1)



CREATE TABLE: Syntax (2)

Column Definition:



Default Column Values (1)

- In the command for creating new table rows (`INSERT INTO`, see below), users do not have to define 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.
- Of course, it is also possible to ensure that a column value must be explicitly defined: `“NOT NULL”` and no `DEFAULT`.

Default Column Values (2)

- E.g. if a new instructor is stored, and no phone number is given, one can automatically insert the department number:

```
CREATE TABLE FACULTY(  
    INAME VARCHAR(20) PRIMARY KEY,  
    PHONE NUMERIC(7) DEFAULT 6249400)
```

- With “DEFAULT SYSDATE” one can put 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”.

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.

- 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, so e.g. the username is always stored as defined in the DEFAULT-clause.

Default Column Values (4)

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

Column Constraints (1)

- In SQL, constraints can be defined as “column constraints” or “table constraints” .
- Column constraints are constraints that refer to only one column. Table constraints 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 constraints are translated to table constraints.

Column Constraints (2)

- Column constraints are specified in the column definition, table constraints are separated by a comma from each other and the column definitions.
- In the example, “NOT NULL” and “REFERENCES *T*” are column constraints, “PRIMARY KEY(SSN, CRN)” is a table constraint:

```
CREATE TABLE ENROLLMENTS(  
    SSN CHAR(11) NOT NULL REFERENCES STUDENTS,  
    CRN NUMERIC(5) NOT NULL REFERENCES COURSES,  
    PRIMARY KEY (SSN, CRN))
```

Column Constraints (3)

- 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 (4)

- PRIMARY KEY implies NOT NULL.

However, DB2 requires that NOT NULL is explicitly specified in addition.

- 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.

- 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).

Column Constraints (5)

- Oracle, SQL Server, and DB2 all exclude subqueries in CHECK-constraints.
- The SQL-92 standard permits them, but then constraint checking is difficult/inefficient.
- 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.

Column Constraints (6)

- A constraint can optionally be given a name by prefixing it with “CONSTRAINT <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, NOT NULL constraints cannot be named.

- Always define a name (except for NOT NULL constraints), otherwise the system chooses names like “SYS_C036”.

Column Constraints (7)

- 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.

Column Constraints (8)

- 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 (9)

```
CREATE TABLE COURSES(  
  CRN          NUMERIC(5)  
              CONSTRAINT COURSES_KEY PRIMARY KEY  
              CONSTRAINT CRN_POSITIVE CHECK(CRN > 0),  
  TOPIC        NUMERIC(4)  
              CONSTRAINT TOPIC_DEFINED NOT NULL  
              CONSTRAINT TOPIC_POS CHECK(TOPIC > 0),  
  TITLE        VARCHAR(40) NOT NULL,  
  INSTRUCTOR   VARCHAR(20)  
              CONSTRAINT INST_REF REFERENCES FACULTY,  
  SEATS        NUMERIC(4)  
              CONSTRAINT VALID_SEATS CHECK(SEATS>=0))
```

Column Constraints (10)

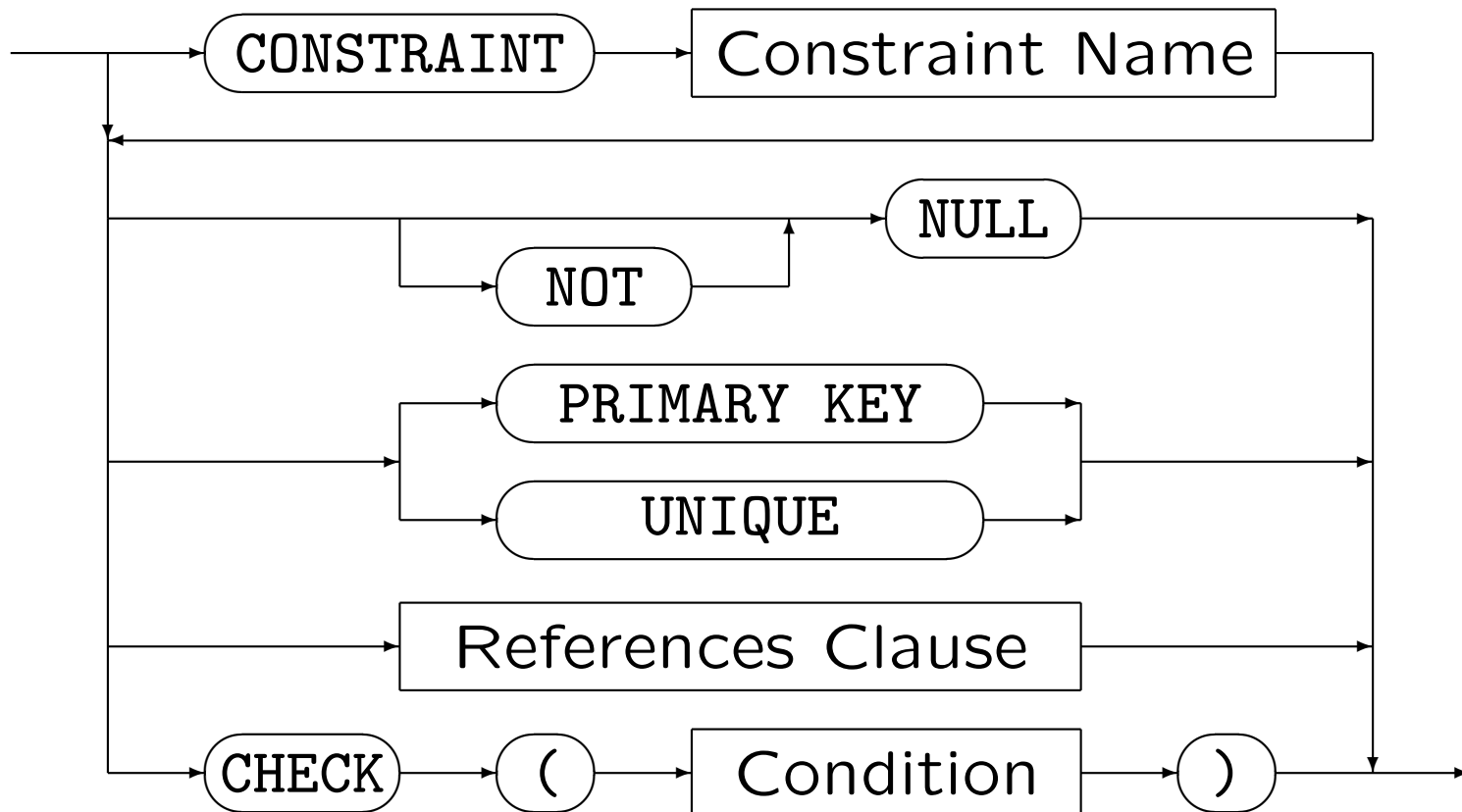


Table Constraints (1)

- 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.

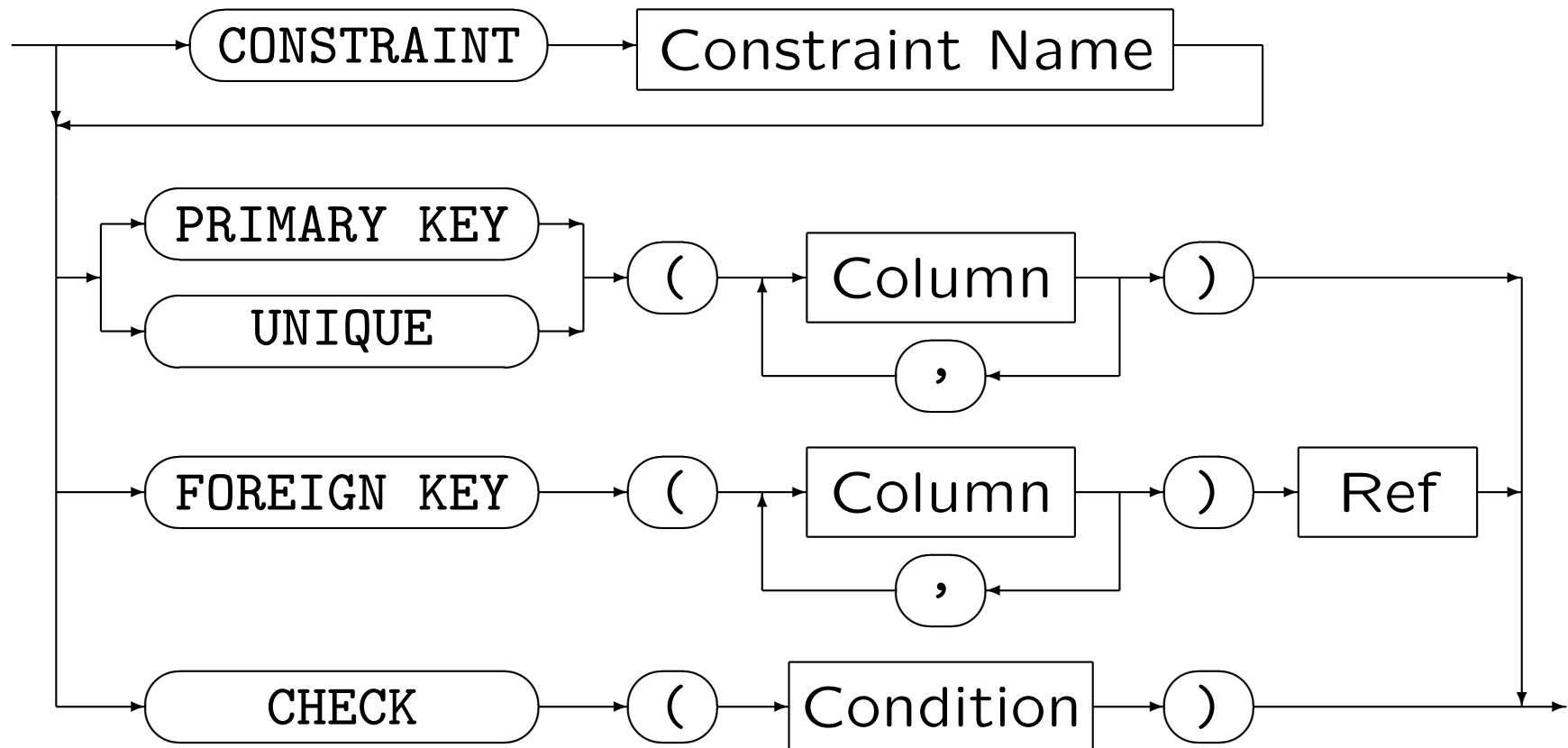
Constraints referring only to one column can be defined as either table constraints or column constraints.

- PRIMARY KEY(A_1, \dots, A_2)
- UNIQUE(A_1, \dots, A_2)
- FOREIGN KEY(A_1, \dots, A_2) REFERENCES R
- CHECK(C)

Table Constraints (2)

```
CREATE TABLE COURSES(  
  CRN          NUMERIC(5)  NOT NULL,  
  TOPIC        NUMERIC(4)  NOT NULL,  
  TITLE        VARCHAR(40) NOT NULL,  
  INSTRUCTOR   VARCHAR(20) NULL,  
  SEATS        NUMERIC(4)  NULL,  
  CONSTRAINT COURSES_KEY PRIMARY KEY (CRN),  
  CONSTRAINT INST_REF FOREIGN KEY (INSTRUCTOR)  
    REFERENCES FACULTY,  
  CONSTRAINT CRN_POSITIVE CHECK(CRN > 0),  
  CONSTRAINT TOPIC_POS CHECK(TOPIC > 0),  
  CONSTRAINT VALID_SEATS CHECK(SEATS >= 0))
```

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.
- It is possible to specify the referenced column names, e.g. `REFERENCES FACULTY(NAME)`.

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 FACULTY ON DELETE CASCADES`

- Otherwise the system rejects attempts to delete an instructor who still has courses in the DB.

References Clause (3)

- In SQL-92 and DB2 (not in Oracle/SQL Server), one can specify that if an instructor 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 three systems.

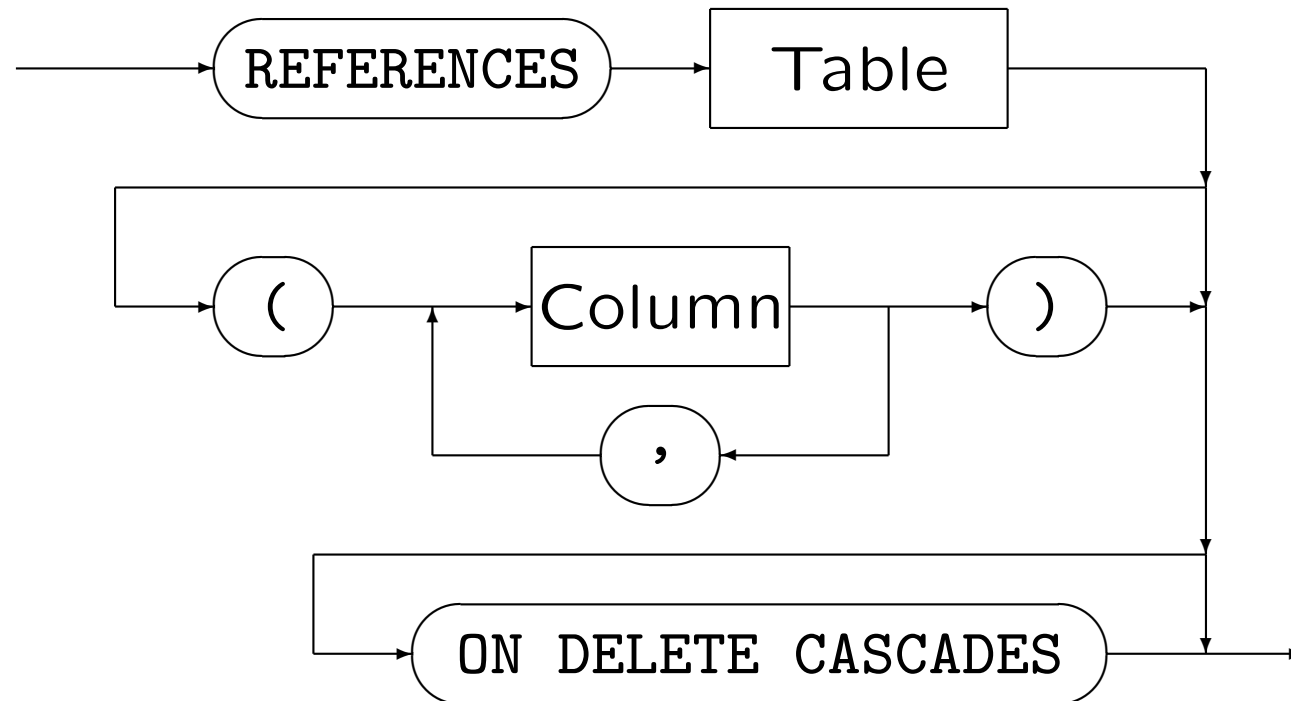
References Clause (4)

- 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.

References Clause (5)



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 (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, 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 statement fails, all are rolled back (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.

Deleting Tables (1)

- Tables can be deleted with the command

```
DROP TABLE <Table Name>
```

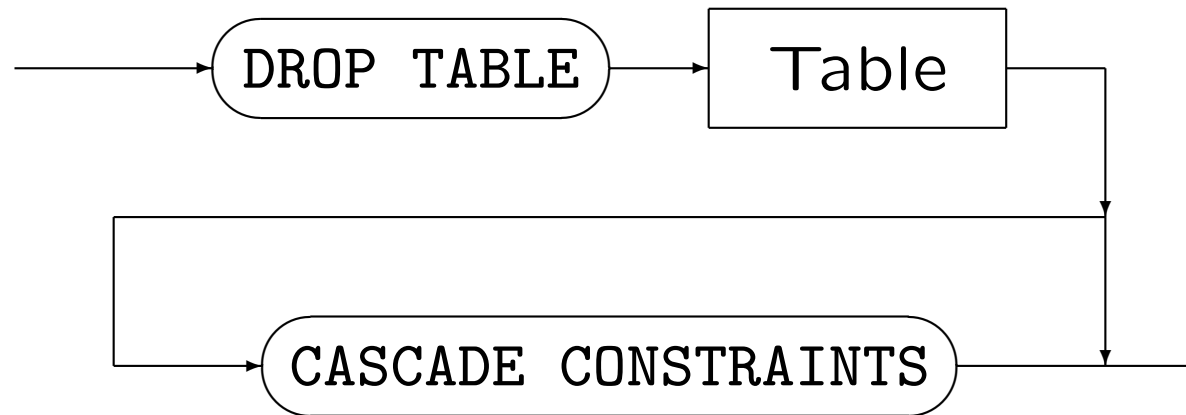
- Of course, this also deletes all rows in the table.

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, you must first drop the referencing table.
 - ◇ 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 ENROLLMENTS**

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

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.
 - ◇ The 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 “ROOM” to the table “FACULTY”:

```
ALTER TABLE FACULTY  
ADD COLUMN ROOM NUMERIC(3)
```
- 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 they were inserted.

ALTER TABLE (5)

SQL-92 (Changing Columns):

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

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

- A column can be removed from a table:

```
ALTER TABLE FACULTY DROP COLUMN Room
```

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

```
ALTER TABLE ACCOUNT ALTER COLUMN CREDIT_LIMIT  
SET DEFAULT 100
```

ALTER TABLE (6)

SQL-92 (Modifying Constraints):

- Add a constraint:

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

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

- Remove a named constraint:

```
ALTER TABLE FACULTY
    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 (7)

Oracle (Adding Columns):

- E.g. add a column “CLOSED” to the table “COURSES”:

```
ALTER TABLE COURSES ADD  
(CLOSED CHAR(1) CHECK(CLOSED IN ('Y','N')))
```

- 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 COURSES ADD  
(CLOSED CHAR(1) DEFAULT 'N' NOT NULL  
CHECK(CLOSED IN ('Y','N')))
```

ALTER TABLE (8)

Oracle (Modifying Columns):

- The datatype of a column can be modified:

```
ALTER TABLE COURSES MODIFY (TITLE 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 (9)

Oracle (Adding/Dropping Constraints):

- Add a constraint:

```
ALTER TABLE COURSES ADD  
    (CONSTRAINT ALTKEY UNIQUE(TITLE))
```

- 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 COURSES 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 (10)

Oracle (Dropping Constraints, Continued):

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

```
ALTER TABLE COURSES DROP PRIMARY KEY CASCADE  
ALTER TABLE COURSES DROP UNIQUE(TITLE)
```

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

```
ALTER TABLE COURSES MODIFY (TITLE 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 (11)

DB2 (Adding Columns):

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

```
ALTER TABLE FACULTY
      ADD COLUMN ROOM NUMERIC(3)
```

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

ALTER TABLE (12)

DB2 (Modifying Columns):

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

```
ALTER TABLE COURSES
```

```
ALTER TITLE 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 (13)

DB2 (Adding/Removing Constraints):

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

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

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

```
ALTER TABLE COURSES DROP CONSTRAINT ROOM_DEFINED
```

- You can drop a primary key without name:

```
ALTER TABLE COURSES DROP PRIMARY KEY
```

ALTER TABLE (14)

SQL Server (Adding/Removing Columns):

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

```
ALTER TABLE FACULTY 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 FACULTY DROP COLUMN ROOM
```

ALTER TABLE (15)

SQL Server (Modifying Columns):

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

```
ALTER TABLE COURSES
```

```
ALTER COLUMN TITLE VARCHAR(100)
```

Decreases in size are possible if the data still fits.

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

```
ALTER TABLE COURSES
```

```
ALTER COLUMN TITLE VARCHAR(100) NULL
```

ALTER TABLE (16)

SQL Server (Adding/Removing Constraints):

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

```
ALTER TABLE FACULTY
    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 COURSES DROP CONSTRAINT ROOM_DEFINED
```

(There is no RESTRICT/CASCADE.)