Objectives

After completing this chapter, you should be able to:

1. Data Types
   - Enumerate the general kinds of data types to expect in a DBMS.
   - Explain the VARCHAR and NUMERIC data types in some detail.
   - Write a CREATE TABLE statement, including keys, foreign keys, and CHECK constraints.
   - Modify the database state with INSERT, UPDATE, and DELETE commands.
   - Explain the various and numerous data types in some detail.

Overview

1. Data Types
   - Data types do not define the set of possible values, but also operations on the values.
   - Data types define not only the set of possible values, but also operations on the values.
   - Different DBMS products support a wide variety of data types.
   - Modern systems allow user-defined data types.

Data Types

The relational model does not depend on a specific selection of data types. Different DBMS products support a wide variety of data types, although strings and numbers of different lengths are always available. Modern systems allow user-defined data types. DB2, Oracle, and SQL Server have some support for user-defined types.

Different DBMS products support a wide variety of data types. Modern systems allow user-defined data types. DB2, Oracle, and SQL Server have some support for user-defined types.

Datatypes define not only the set of possible values, but also operations on the values. E.g., +, -, * for numbers, / for strings.

See Appendix for lists of data type operations.

References

Chap. 8, "SQL: The Relational Database Standard".

Chapter 4: "SQL", Section 6.2: "Referential Integrity".

Kemper/Eickler: Datenbanksysteme (in German), Ch. 4, Oldenburg, 1997.

Lippich: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.


van der Lans: SQL, Der ISO-Standard (in German, there is an English version), Hanser, 1990.


Microsoft SQL Server Books Online: Accessing and Changing Data.
Character Strings

**Character Strings (1)**

Characters are not so clear.

- **Categories of Data Types:**
  - Relatively standardized:
    - Character strings (fixed length, variable length)
    - Numbers (integer, fixed point, floating point)
    - Supported, but differently in each DBMS:
      - Numbers (integer, fixed point, floating point)
      - Character strings (fixed length, variable length)
      - Relatively standardized:
    - Binary data
    - National language character strings

**Categories of Data Types:**

- **Character Strings (1):**
  - CHARACTER (n):
    - Fixed length string of n characters.
    - Data which is stored in a column of this type is filled with spaces to the defined length n.
    - CHARACTER can be abbreviated to CHAR.
    - If no length is specified, 1 is assumed.
    - This data type was already contained in the SQL-68 standard.
    - Of course, Oracle, SQLServer, DB2 support it.
    - In Oracle, the size must be between 1 and 255.
    - The maximum size is 8000 in SQL Server, and 254 in DB2.
    - The data type was already contained in the SQL-86 standard.
    - The data type allows storage of single characters.
    - So the space for a character is always needed.
    - Fixed length strings of n characters.

- **Character Strings (2):**
  - VARCHAR (n):
    - Variable length string of up to n characters.
    - Like user-defined and DBMS-specific data types:
    - Date and time values
    - National language character strings
    - Binary data
    - Long character data
    - Supported, but differently in each DBMS:
      - Character strings (fixed length, variable length)

**Character Strings (2):**
The default installation/settings in the three DBMS are:

- **Oracle**: All uppercase characters come before all lowercase characters, e.g., 'Z' < 'a'.
- **DB2**: Upper case and lower case characters are interleaved, e.g., 'a' < 'A' < 'b'.
- **SQL Server**: By default case-insensitive, e.g., 'a' = 'A'.

It might be possible to change this, but only during installation/during database creation.

The shorter string is held with '>', before the comparison.

DB2 and SQL Server seem to use always the blank-padded semantics (least by default).

- **In DB2**, the non-padded semantics is used at last one expansion of a comparison operator has type VARCHAR.
- **In Oracle**, the non-padded semantics is used at least one character.
- **In DB2** and SQL Server seem to use always the blank-padded semantics (least by default).

The system compares character by character, the first comparison which does not give '=' determines the result.

- **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.
- **Non-padded Comparison Semantics**: E.g., 'a' < 'A'.
- **Blank-Padded Comparison Semantics**: E.g., 'a' = 'A'.

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'.
  - **Blank-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.
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 $s \leq p$. Negative scales generate zeros at the end. Scales which are greater than the precision generate zeros before the stored digits.

Oracle normally uses NUMBER($p$, $s$) and NUMBER($p$) but understands NUMERIC($p$, $s$) and DECIMAL($p$, $s$) as synonyms.

Neither DB2 nor SQL Server understands NUMBER.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 15 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 14 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 13 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 12 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 11 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 10 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 9 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 8 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 7 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 6 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 5 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 4 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 3 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 2 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 1 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 0 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 0 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 0 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 0 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 0 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.

SMALLINT is as SMALLINT as INTEGER, but possibly a smaller range of values.

DB2 and SQL Server use 0 bit binary numbers.

Oracle treats INTEGER as a synonym for NUMBER($p$).

INTEGER can be abbreviated INT.
Data Types in SQL

- **CHAR**
  - \([\text{ACTER}\(n\)]\)
  - Mark optional parts.

- **NUMERIC**
  - \([\text{IMAL}\(p, s\)]\)
  - Decimal numbers.
  - Plus long character data.

- **INT**

- **SMALLINT**

- **FLOAT**
  - \([\text{REAL}\(p\)]\)

- **DOUBLE PRECISION**

These types should be very portable. All three systems understand them (plus VARCHAR).

- **CHARACTER LARGE OBJECT** (LONG VARCHAR)
  - Used for long values.
  - A table can have at most one column used for long values. A table can have at most one column.
  - \(\ldots\) LIKE, BETWEEN, and other string functions cannot be used for LONG values.
  - At a table can have at most one column.

- **LONG VARCHAR** (up to 2GB or data).
  - If a longer text has to be searchable with LIKE, it must be split into lines or paragraphs, stored separately as VARCHAR.

- **SET LONG** in data types/strings, stored separately as VARCHAR.

- **DB2** has character data in DB2.

- **Oracle**

- **CLOB**
  - Character large object, up to 4GB (new in Oracle 8/9).
  - This is something like a file, stored inside the database, with its own identity (LOB locator). It is very similar to LONG.
  - The programmatic interface allows random access to the data.

- **LONG**
  - Character strings of up to 2GB length.

- **DB2** has character data in DB2.
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 2GB - 1 = 2,147,483,647.

TEXT columns are also not allowed as arguments to string functions such as concatenation.

There are also binary large objects, BLOBs, which can hold up to 2GB of binary data.

There are also binary large objects, BLOBs, which can hold up to 2GB of binary data.

OFFICE VARCHAR(100) FOR BIT DATA

String columns can be declared as containing binary data:

- BIT
- BIT VARYING
- LONG RAW
- BLOB (n)

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

Whenever the meaning of the data is predetermined by an external program and not known to the database, comparison is done literally (binary codes).

String constants can be written in the form X/'C/5'/.

There are no built-in functions which work with TEXT.

Input/Output as character strings with hexadecimal digits.

There are also binary large objects, BLOBs, which can hold up to 2GB of binary data.

Stefan Brass

Binary Data in Oracle

CHAR (1) is used together with a constraint that column has a boolean data type.

Accordingly, neither the SQL-92 standard nor one of the three DBMSs supports these columns.

2GB is 2,147,483,647.

String constants can be written in hexadecimal notation, e.g., B/'1/1/0/0/1/0/1'/.

TEXT columns are also not allowed as arguments to string functions such as concatenation.

CONCATENATION, but there are some database functions such as "WHAT WORKS WITH TEXT".
**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 by default.
  - Constants are written in the form \(^{0xFF}1C\).

- **VARBINARY**(n)**:** Variable-length binary data of up to n bytes.
  - The size n can be at most 8000.
  - Input data is filled with 0x00 by default.
  - Constants are written in the form \(\text{DATE}/'1965-03-27/00:00:00/'\).

**Date/Time Types in SQL Server**

- **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 1900-01-01 are excluded.
  - DATE constants are written as a character string of the form YYYY-MM-DD.

- **TIME**:** A time of day (between 00:00:00 and 23:59:59).
  - 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.

- **TIMESTAMP**:** DATE and TIME together.
  - For example: \(\text{TIMESTAMP}/'1965-03-27/00:00:00/'\).

**INTERVAL DAY(p)**:** Period of time in days.
  - \(0 < p < n \leq 9999\) days.

- **INTERVAL HOUR(p) TO SECOND**:** Difference between two TIME values.

**DB2** supports DATE, TIME, and TIMESTAMP.

**DATE/Time Types in DB2**

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

**DB2** supports DATE, TIME, and TIMESTAMP.

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

- **TIME**:** A time of day (between 00:00:00 and 23:59:59).
  - 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.

- **TIMESTAMP**:** DATE and TIME together.
  - For example: \(\text{TIMESTAMP}/'1965-03-27/00:00:00/'\).

- **INTERVAL DAY(p)**:** Period of time in days.
  - \(0 < p < n \leq 9999\) days.

**DB2** supports DATE, TIME, and TIMESTAMP.

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

- **TIME**:** A time of day (between 00:00:00 and 23:59:59).
  - 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.

- **TIMESTAMP**:** DATE and TIME together.
  - For example: \(\text{TIMESTAMP}/'1965-03-27/00:00:00/'\).
National Language Character Sets (2)

Oracle can store modern and traditional Chinese characters.


Oracle can transform characters between different encodings.

Oracle can store special German characters like å, ö, ü.

Oracle has no specific classifications for the data type DATE.

Oracle has a type for timestamps called DATE.

Oracle does not support the SQL-92 date/time types.

National Language Character Sets (1)

The NLS_DATE_FORMAT setting determines the default output format:

In Germany, 'DD.MM.YY' is used instead of 'DD-MON-YY'.

In China, Chinese characters are used.

Oracle can display error messages etc. in different languages.

Oracle can print error messages etc. in different environments.

Oracle supports different character encodings.

Oracle can work with Japanese characters.

Oracle can store special German characters like å, ö, ü.
Overview

1. Data Types

CREATE TABLE Course (CRN NUMERIC(5) NOT NULL CHECK (CRN > 0),
Title VARCHAR(40) NOT NULL,
IName VARCHAR(20) NOT NULL,
CONSTRAINT CK_CRN CHECK (CRN > 0),
CONSTRAINT PK_Course PRIMARY KEY (CRN))

Example

Other Data Types (1)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWIDs are used to access a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Other Data Types (2)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

DB2:
- CREATE TABLE Course (CRN NUMERIC(5) NOT NULL CHECK (CRN > 0),
Title VARCHAR(40) NOT NULL,
IName VARCHAR(20) NOT NULL,
CONSTRAINT CK_CRN CHECK (CRN > 0),
CONSTRAINT PK_Course PRIMARY KEY (CRN))

Oracle:
- CREATE TABLE Course (CRN NUMERIC(5) NOT NULL CHECK (CRN > 0),
Title VARCHAR(40) NOT NULL,
IName VARCHAR(20) NOT NULL,
CONSTRAINT CK_CRN CHECK (CRN > 0),
CONSTRAINT PK_Course PRIMARY KEY (CRN))

SQL Server:
- CREATE TABLE Course (CRN NUMERIC(5) NOT NULL CHECK (CRN > 0),
Title VARCHAR(40) NOT NULL,
IName VARCHAR(20) NOT NULL,
CONSTRAINT CK_CRN CHECK (CRN > 0),
CONSTRAINT PK_Course PRIMARY KEY (CRN))

Other Data Types

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Example

Other Data Types (1)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Other Data Types (2)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Example

Other Data Types (1)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Other Data Types (2)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Example

Other Data Types (1)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Other Data Types (2)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Example

Other Data Types (1)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Other Data Types (2)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Example

Other Data Types (1)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Other Data Types (2)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Example

Other Data Types (1)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Other Data Types (2)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.

Example

Other Data Types (1)

- ROWID: Reference to a physical row.
- ROWID components are shown.
- ROWID is a 32-bit number.
- ROWID has a dedicated column "ROWID".
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
- ROWID can be used to locate a specific row.
- ROWID is a pointer to a specific row.
- ROWID is used to locate a specific row.
Default Values for Columns (2)

- In DB2, use `CURRENT TIMESTAMP`.
- In SQL Server and SQL-92, use `CURRENT_TIMESTAMP`.
- `USER` was already contained in the SQL-88 standard and should be freely usable.
- `SYSDATE` was added specifically for Oracle.
- `CURRENT_TIMESTAMP` works only in Oracle.

Database users can be given selective `INSERT` rights, so that
- users do not have to define values for all columns.
- Users can automatically insert the department number.
- E.g. if a new instructor is stored, and no phone number is stored, the department number should be stored in columns for which no value was given.
- However, it is possible to specify a default value which
- Usually a null value is put in the missing columns.
- In the command for creating new table rows (see below),

Default Values for Columns (1)

CREATE TABLE Instructor

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Column Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR(20)</td>
<td>PRIMARY KEY</td>
</tr>
<tr>
<td>NUMERIC(7)</td>
<td>DEFAULT 6249400</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Table Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE</td>
<td>Instructor</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In DB2, NOT NULL constraints cannot be named.

In DB2, constraint names must be unique only in a table. Column names may have to be unique within a database.

Column names must be unique in the schema, where a constraint is defined.

A constraint can optionally be given a name by prefixing it with CONSTRAINT (name).

A constraint can optionally be given a name by prefixing it with CONSTRAINT (name).

In Oracle, any term without a column name is allowed.

- CHECK (cond): Values in this column must satisfy (cond).
- REFERENCES (Table): The column is a foreign key, values must appear in the primary key column of (Table).
- PRIMARY KEY: This column is the primary key of the table. The column is not null, and uniqueness does not imply NOT NULL.
- NOT NULL: The column must be defined (not a null value).

SQL Server supports only one CHECK constraint per column.

SQL Server supports only one CHECK constraint per column.

- CHECK (cond): Values in this column must satisfy (cond).
- REFERENCES (Table): The column is a foreign key, values must appear in the primary key column of (Table).
- PRIMARY KEY: This column is the primary key of the table. The column is not null, and uniqueness does not imply NOT NULL.
- NOT NULL: The column must be defined (not a null value).

In DB2, constraint names must be unique only in a table. Column names may have to be unique within a database.

Column names must be unique in the schema, where a constraint is defined.

A constraint can optionally be given a name by prefixing it with CONSTRAINT (name).

A constraint can optionally be given a name by prefixing it with CONSTRAINT (name).

In Oracle, any term without a column name is allowed.

- CHECK (cond): Values in this column must satisfy (cond).
- REFERENCES (Table): The column is a foreign key, values must appear in the primary key column of (Table).
- PRIMARY KEY: This column is the primary key of the table. The column is not null, and uniqueness does not imply NOT NULL.
- NOT NULL: The column must be defined (not a null value).

In DB2, constraint names must be unique only in a table. Column names may have to be unique within a database.

Column names must be unique in the schema, where a constraint is defined.

A constraint can optionally be given a name by prefixing it with CONSTRAINT (name).
Integrity constraints count as valid if they result in the truth value NULL. Constraints referring only to one attribute or a CHECK condition on more than one attribute or a CHECK constraint on a key or a foreign key are needed when a key or a foreign key is not NULL.

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

Column constraints are separated by a comma from column definitions and each other.

SQL III/11

Column constraints (1)

Column constraints (4)

Column constraints (5)

Table constraints (1)

Table constraints (5)

By an optional Integrity Enhancement Feature (SQL-89), enforcement of constraints is in 1988, the standard was extended to enforce key constraints. In 1999, the standard was extended and particular was not implemented in many systems.

In 1988, only CHECK [NOT NULL [UNIQUE]] was often used to enforce attributes of which only some are null or mandatory.

In 1989, NULL was not supported.

The checking for keys and foreign keys which consist of multiple attributes, of which only some are null or mandatory, was not supported.

Table constraints (5)

Column constraints (6)
The effects of updates on the key of the instructor (name change):

- In SQL-92, one may specify "SET DEFAULT" and "SET NULL" in the DB, but their attributes are not set to null when an instructor is deleted, hence:
  - In SQL-92 and DB2, the instructor may still have courses in the DB.
  - Otherwise, the system rejects attempts to delete an instructor that has courses in the DB.

REFERENCES instructor ON DELETE CASCADE:

One can request (in SQL-92, Oracle, DB2, not in SQL Server):

- If an instructor is deleted, his/her courses are deleted, too.
- REFERENCES Instructor ON DELETE CASCADE.

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

REFERENCES Constraint (name change):

- It is possible to specify the referenced column names.
- In table constraints, e.g., REFERENCES Constraint, a REFERENCE Constraint is used after the column name in the column constraint.

Table Constraints (1):

- CONSTRAINT Course (CRN): CHECK (CRN > 0).
  - CONSTRAINT Course (CRN) PRIMARY KEY.
  - CONSTRAINT Course (CRN) NOT NULL.
  - TITLE VARCHAR(200) NOT NULL.
  - CONSTRAINT Course (CRN) CHECK (CRN > 0).

Table Constraints (2):

- REFERENCES Instructor ON DELETE CASCADE.
  - CONSTRAINT Instructor (IName) PRIMARY KEY.
  - CONSTRAINT Instructor (IName) NOT NULL.
  - CONSTRAINT Course (IName) FOREIGN KEY (CRN) REFERENCES Instructor (IName) ON DELETE CASCADE;
If a row must be stored in multiple blocks:
• The row length is not limited, but the performance degrades
  unreachable when the database is created.

The database block size can be configured within
  block size.
• Key values (or indexed values) are limited to 40% of the
  max. 32 columns per key.
• max. 1000 columns per table.

Oracle:

CREATE TABLE: Restrictions (2)

SQL Server:

• Keys can contain max. 16 columns and max. 255 bytes.
  The data of TEXT etc. columns are stored separately.
  max. 800 bytes per row
  max. 500 columns per table

DB2:

• Keys can contain max. 16 columns and max. 900 bytes.
  The data of TEXT etc. columns are stored separately.
  max. 1024 columns per table

SQL Server:

CREATE TABLE: Restrictions (3)

Oracle:

ON DELETE CASCADES

REFERENCES

Column

Table

References Clause (3)
CREATE SCHEMA (Name) AUTHORIZATION (User)

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

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

If any of the statements fail, all are rolled back (undone).

In DB2, the CREATE VIEW and CREATE INDEX is allowed.

In DB2, schemas and users are separate things, although the schema name is the authorization clause.

In Oracle, the statement fails, all are rolled back (undone).

If the table is referenced in foreign key constraints,

In SQL Server, you must first drop the referencing table.

In Oracle, CASCADE CONSTRAINTS can be specified to drop the foreign keys, too.

What if the table is referenced in foreign key constraints?

In DB2, the foreign key is automatically dropped.

In SQL Server, you must first drop the referenced table.

In SQL-92, it is only "CASCADE"

Transactions: So long as possible for the action.

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

DROP TABLE (table_name)

Tables can be deleted with the command

Deleting Tables (1)
Overview

1. Data Types
2. Table Definition
3. Update Commands
4. Alter Table

Update Commands

SQL has three commands to change the DB state:

- ROLLBACK: Transaction failed; undo all changes.
- COMMIT: Transaction successful; make changes durable.

In addition, SQL has two commands for ending transactions:
- DELETE: For deleting rows from a table.
- UPDATE: For changing attribute values of existing rows.
- INSERT: For inserting new rows into a table.

INSERT

The default value is inserted in place of NULL:

\[
\text{INSERT INTO } \text{Instructor} \text{VALUES (Brass, 9404)}
\]

If it is possible to specify values for only a subset of the columns:

 DEFAULT, and any term \{NAME, "PASS", "SASDIE", \}

Not only constants are allowed as values, but also \text{NULL}.

First form:

- For inserting the result of a query into another table.
- For inserting a single row with new data elements.

The INSERT command has two forms:

\[
\text{INSERT INTO } \text{Instructor (IName) VALUES (Brass, 9404)}
\]
SQL III

**Example:**

```
INSERT INTO My_Courses(CRN, Title, Term)
SELECT CRN, Title, 'Fall 2023'
FROM Course
WHERE IName LIKE '%Brass/%'
```

Therefore, it is much faster.

In contrast to `DELETE`, it cannot be rolled back (undone).

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.

Oracle and SQL Server (not DB2) have a command `TRUNCATE TABLE` 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.
The `UPDATE` command is for changing attribute values of selected tuples.

**Example:**
```sql
UPDATE Emp
SET Sal = Sal * 1.1
WHERE Deptno = 20
```

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

**Example:**
```sql
UPDATE Emp
WHERE Deptno = 20
SET Sal = 1.1 * Sal
```

**Examples:**
- All employees from department 20: 10% raise.
- The `UPDATE` command is for changing attribute values of.
If a table is large, this might be a very costly operation.

The work is then saved back into the database.

If a table is small, it is usually easier to drop the table and

If there should be a power failure etc., only the changes

After the last COMMIT or ROLLBACK, attempts to read transaction management

There is much more to say about transaction management

In the SQL*Plus it is possible to set autocommit off, which

If the last COMMIT was rolled back, the changes

If one works with the database for a long time, one should COMMIT the work from time to time.
ALTER TABLE (Oracle)
**ALTER TABLE ALTER COLUMN** in SQL Server (2)

- You can change the data type of a column, but not its size.
- You can add a constraint (either a check constraint or a primary key or a foreign key).
- You can drop a column (there is no requirement that a column cannot be dropped).
- If you specify a default value, the column can be not null.
- You can add a column to a table, but not to a view.
- You can add a column to a table, but not to a view.

**ALTER TABLE ALTER COLUMN** in SQL Server (1)

- You can alter the data type of a column, but not its size.
- You can add a constraint (either a check constraint or a primary key or a foreign key).
- You can drop a column (there is no requirement that a column cannot be dropped).
- If you specify a default value, the column can be not null.
- You can add a column to a table, but not to a view.
- You can add a column to a table, but not to a view.

**ALTER TABLE ALTER COLUMN** in SQL Server (3)

- You can drop a column (there is no requirement that a column cannot be dropped).
- If you specify a default value, the column can be not null.
- You can add a column to a table, but not to a view.
- You can add a column to a table, but not to a view.

**ALTER TABLE ALTER COLUMN** in SQL Server (4)

- You can alter the data type of a column, but not its size.
- You can add a constraint (either a check constraint or a primary key or a foreign key).
- You can drop a column (there is no requirement that a column cannot be dropped).
- If you specify a default value, the column can be not null.
- You can add a column to a table, but not to a view.
- You can add a column to a table, but not to a view.