ECE30030/ITP30010 Database Systems

SQL DDL

Reading: Chapter 3

Charmgil Hong

charmgil@handong.edu

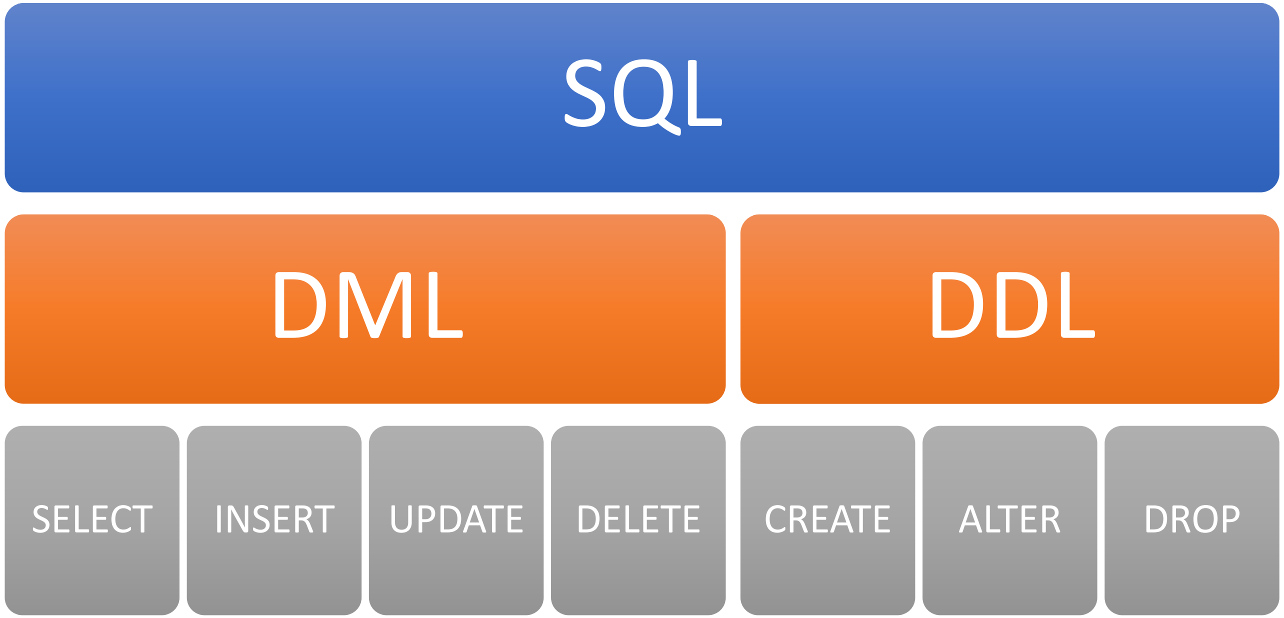
Spring, 2025

Handong Global University

Agenda

- SQL DDL (Data Definition Language)

SQL Commands



Data Definition Language

- The SQL data-definition language (DDL) allows the specification of information about relations, including:
  - The schema for each relation
  - The type of values associated with each attribute
  - The Integrity constraints
  - The set of indices to be maintained for each relation
  - Security and authorization information for each relation
  - The physical storage structure of each relation on disk

Domain Types in SQL

- SQL Data Types
  - **CHAR(*n*)**: Fixed length character string, with user-specified length *n*
    - Maximum length *n* = [0, 255]
  - **VARCHAR(*n*)**: Variable length character strings, with user-specified maximum length *n*
    - Maximum length *n* = [0, 65,535]

    - *If the length is always the same, use a CHAR-type attribute;*
      if you are storing wildly variable length strings, use a VARCHAR-type attribute

  - **TEXT**: for strings longer than the range of VARCHAR

TINYTEXT	0 – 255 bytes
TEXT	0 – 65,535 bytes
MEDIUMTEXT	0 – 16,777,215 bytes
LONGTEXT	0 – 4,294,967,295 bytes

Domain Types in SQL

- Difference between CHAR and VARCHAR

Value	CHAR(4)	Storage	VARCHAR(4)	Storage
‘’	‘ ‘	4 bytes	‘’	1 bytes
‘ab’	‘ab ‘	4 bytes	‘ab’	3 bytes
‘abcd’	‘abcd’	4 bytes	‘abcd’	5 bytes
‘abcdefg’	‘abcd’	4 bytes	‘abcd’	5 bytes

Domain Types in SQL

- “\”%ab%\””

Domain Types in SQL

- SQL Data Types
  - **INT, INTEGER**: Integer (a finite subset of the integers that is machine-dependent)
  - **SMALLINT**: Small integer (a machine-dependent subset of the integer domain type)
  - **BIGINT**: Small integer (a machine-dependent subset of the integer domain type)

  - **TINYINT** and **MEDIUMINT** are also available

Domain Types in SQL

- Different R-DBMSs support different combinations of those integer types

	Bytes	MySQL	MS SQL	PostgresSQL	DB2
TINYINT	1	✓	✓		
SMALLINT	2	✓	✓	✓	✓
MEDIUMINT	3	✓			
INT/INTEGER	4	✓	✓	✓	✓
BIGINT	8	✓	✓	✓	✓

  - *C.f.*, Oracle only has a NUMBER datatype

Domain Types in SQL

- SQL Data Types
  - **NUMERIC(*p*,*d*)**: Fixed point number (exact value) with user-specified precision of *p* digits, with *d* digits to the right of decimal point
    - *E.g.*, **NUMERIC**(3,1) allows 44.5 to be stores exactly, but not 444.5 or 0.32)
    - In MySQL, **DECIMAL** is NUMERIC
  - **FLOAT**: Floating point number (approximate) with single-precision
  - **REAL, DOUBLE**: Floating point number (approximate) with double-precision

Domain Types in SQL

- DECIMAL vs INT/FLOAT/DOUBLE
  - FLOAT and DOUBLE are faster than DECIMAL
  - DECIMAL values are exact
    - Example

floats: FLOAT	decimals: DECIMAL(3,2)
1.1	1.10
1.1	1.10
1.1	1.10

- SELECT SUM(…) → DECIMAL values are precise

SUM(floats)	SUM(decimals)
3.3000000715255737	3.30

Domain Types in SQL

- SQL Data Types
  - **DATE**: ‘YYYY-MM-DD’
    - Rage: 1000-01-01 to 9999-12-31
    - *E.g.*, ‘2020-03-01’ for March 1, 2020
  - **TIME**: ‘HH:MM:SS’
    - Range: -838:59:59 to 838:59:59
    - *E.g.*, ’14:30:03.5’ for 3.5 seconds after 2:30pm
  - **DATETIME**: ‘YYYY-MM-DD HH:MM:SS’
    - Range: 1000-01-01 00:00:00 to 9999-12-31 23:59:59
  - **YEAR**: ‘YYYY’
    - Range: 1901 to 2155, or 0000 (illegal year values are converted to 0000)

Domain Types in SQL

- SQL Data Types
  - **TIMESTAMP(*n*)**: Unix time (time since Jan 1, 1970)
    - Range: 1970-01-01 00:00:01 UTC to 2038-01-19 03:14:07 UTC
    - Typically used for logging (keeping records of all the system events)
    - Depending on size *n*, the display pattern changes

	Format
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

Domain Types in SQL

- SQL Data Types
  - **BINARY(*n*)**: binary byte data type, with user-specified length *n*
    - Contains a byte strings (rather than a character string)
    - Maximum length *n* = [0, 255]
  - **VARBINARY(*n*)**: binary byte data type, with user-specified maximum length *n*
    - Maximum length *n* = [0, 65,535]
  - **BLOB**: Binary Large OBject data type

TINYBLOB	0 – 255 bytes
BLOB	0 – 65,535 bytes (65 KB)
MEDIUMBLOB	0 – 16,777,215 bytes (16 MB)
LONGBLOB	0 – 4,294,967,295 bytes (4 GB)

CREATE TABLE Construct

- A new relation is defined using the **CREATE TABLE** command:

  CREATE TABLE *r*
  $(A_1\ D_1, A_2\ D_2, \ldots, A_n\ D_n,$
  $(\textit{integrity-constraint}_1),$
  …,
  $(\textit{integrity-constraint}_k))$

  - *r* is the name of the relation
  - Each A_i is an attribute name in the schema of relation *r*
  - Each D_i is the data type of values in the domain of attribute A_i

- Example:

```
CREATE TABLE instructor(
        ID              CHAR(5),
        name            VARCHAR(20),
        dept_name       VARCHAR(20),
        salary          NUMERIC(8,2))
```

Integrity Constraints in CREATE TABLE

- SQL prevents any update to the database that violates an integrity constraint
  - Integrity constraints allow us to specify what data makes sense for us
- Types of integrity constraints
  - Primary key: **PRIMARY KEY** (A_1, ..., A_n)
  - Foreign key: **FOREIGN KEY** (A_m, ..., A_n) **REFERENCES** *r*
  - Unique key: **UNIQUE**
  - Not null: **NOT NULL**
- Example:

```
CREATE TABLE instructor(
          ID              CHAR(5),
          name            VARCHAR(20) NOT NULL,
          dept_name       VARCHAR(20)
          salary          NUMERIC(8, 2),
          PRIMARY KEY (ID),
          FOREIGN KEY (dept_name) REFERENCES department);
```

Declaring Keys

- An attribute or list of attributes may be declared as PRIMARY KEY or UNIQUE
  - Meaning: no two tuples of the relation may agree in all the attribute(s) on the list
    - That is, the attribute(s) do(es) not allow duplicates in values
    - PRIMARY KEY/UNIQUE can be used as an identifier for each row
  - Comparison: PRIMARY KEY vs UNIQUE

PRIMARY KEY	UNIQUE
Used to serve as a unique identifier for each row in a relation	Uniquely determines a row which is not primary key
Cannot accept NULL	Can accept NULL values (some DBMSs accept only one NULL value)
A relation can have only one primary key	A relation can have more than one unique attributes
Clustered index	Non-clustered index

Declaring Keys

-
```
CREATE TABLE student (
      ID             VARCHAR(5),
      name           VARCHAR(20) NOT NULL,
     dept_name       VARCHAR(20),
      tot_cred       NUMERIC(3,0),
     PRIMARY KEY (ID),
     FOREIGN KEY (dept_name) REFERENCES department);
```

More Examples

```
• CREATE TABLE student (
        ID              VARCHAR(5) PRIMARY KEY,
        name            VARCHAR(20) NOT NULL,
       dept_name        VARCHAR(20),
        tot_cred        NUMERIC(3,0),
       FOREIGN KEY (dept_name) REFERENCES department);
```

More Examples

```
• CREATE TABLE takes (
        ID              VARCHAR(5),
        course_id       VARCHAR(8),
        sec_id          VARCHAR(8),
        semester        VARCHAR(6),
        year            NUMERIC(4,0),
        grade           VARCHAR(2),
        PRIMARY KEY (ID, course_id, sec_id, semester, year),
        FOREIGN KEY (ID) REFERENCES  student,
        FOREIGN KEY (course_id, sec_id, semester, year)
                        REFERENCES section);
```

More Examples

```
• CREATE TABLE course (
        course_id       VARCHAR(8),
        title           VARCHAR(50),
        dept_name       VARCHAR(20),
        credits         NUMERIC(2,0),
        PRIMARY KEY (course_id),
        FOREIGN KEY (dept_name) REFERENCES department);
```

More Examples

-
  ```
  CREATE TABLE course (
          course_id       VARCHAR(8),
          title           VARCHAR(50),
          dept_name       VARCHAR(20) DEFAULT ‘Comp. Sci’,
          credits         NUMERIC(2,0),
          PRIMARY KEY (course_id),
          FOREIGN KEY (dept_name) REFERENCES department);
  ```

More Examples

-
```
CREATE TABLE neighbors(
        name   CHAR(30) PRIMARY KEY,
        addr   CHAR(50) DEFAULT ‘123 Sesame St.’,
        phone  CHAR(16));
```

  - Inserting Elmo is a neighbor:
    - INSERT INTO neighbors (name)
      VALUES (‘Elmo’);

name	addr	phone
‘Elmo’	‘123 Sesame St.’	NULL

More Examples

-
```
CREATE TABLE neighbors(
        name   CHAR(30) PRIMARY KEY,
        addr   CHAR(50) DEFAULT ‘123 Sesame St.’,
        phone  CHAR(16) NOT NULL);
```

  - Inserting Elmo is a neighbor:
    - INSERT INTO neighbors (name)
      VALUES (‘Elmo’);

      → If phone were NOT NULL, this insertion would have been rejected

Table Updates (Updating Tuples)

- INSERT
  - **INSERT INTO** *instructor* **VALUES** (‘10211’, ‘Smith’, ‘Biology’, 66000)

- DELETE
  - **DELETE FROM** *student*
    - Remove all tuples from the *student* relation

Table Updates (Updating Table Schemas)

- DROP TABLE
  - **DROP TABLE** *r*
    - Remove relation *r*

- ALTER
  - **ALTER TABLE** *r* **ADD** *A D*
    - *A* is the name of the new attribute to add to relation *r*; *D* is the domain of *A*
    - All existing tuples in the relation are assigned *null* as the value for the new attribute
  - **ALTER TABLE** r **DROP** *A*
    - *A* is the name of an attribute in *r*
    - Dropping of attributes not supported by many databases (MySQL does)

Table Updates (Updating Table Schemas)

- Examples
  - **DROP TABLE** time_slot_backup;
  - **ALTER TABLE** time_slot_backup **ADD** remark VARCHAR(20);
  - **ALTER TABLE** time_slot_backup **DROP** remark;

EOF

- Coming next:
  - Designing a database