ECE30030/ITP30010 Database Systems

SQL DDL

Reading: Chapter 3

Charmgil Hong

charmgil@handong.edu

Spring, 2023

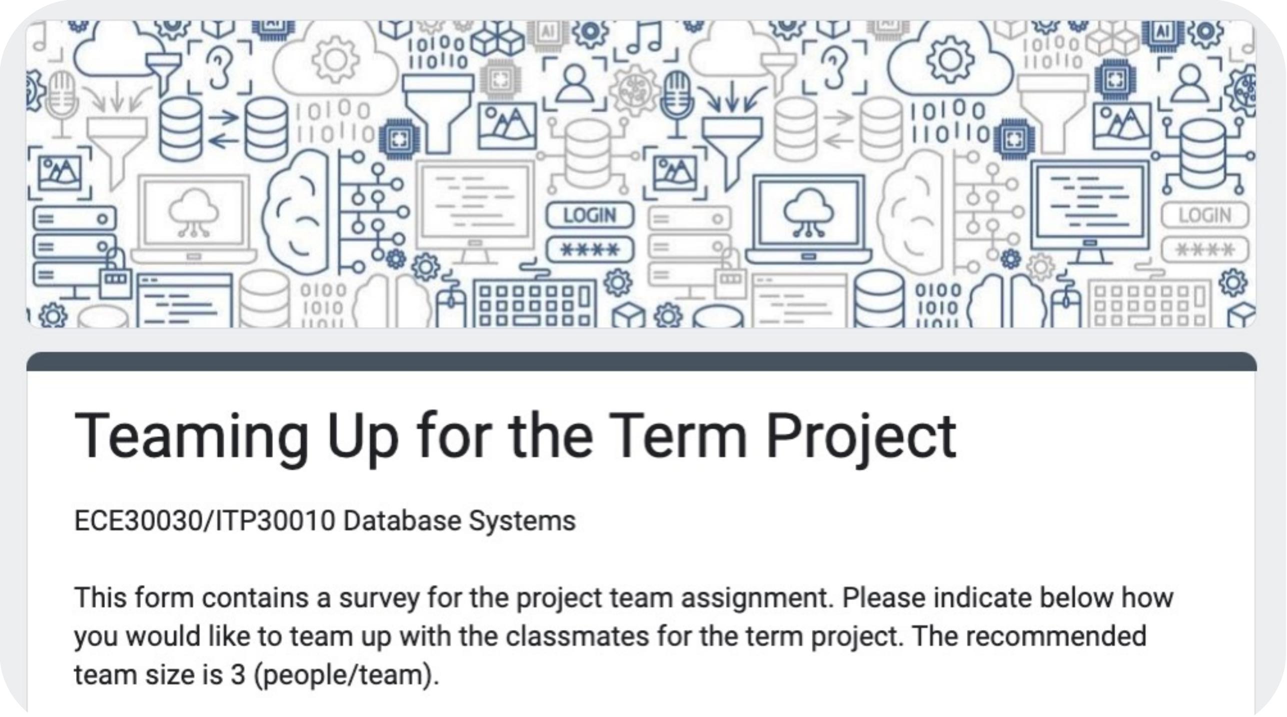
Handong Global University

Announcement

- Homework assignment #3 is out
  - Due: By the end of Saturday, April 22
  - Please start early

Announcement

- Forming teams for the term project
  - Response due: Monday, April 10
  - URL: https://forms.gle/kQWG9ML6fqytYm7p7
  - Problem & data release: Week #8

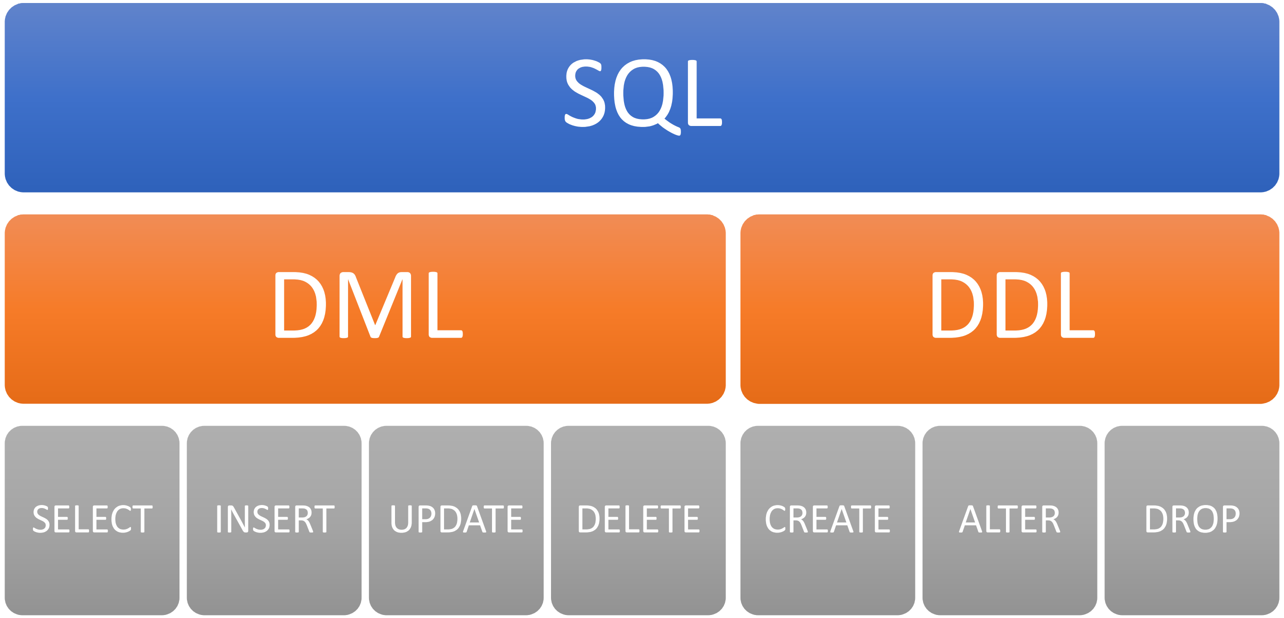


Teaming Up for the Term Project

ECE30030/ITP30010 Database Systems

This form contains a survey for the project team assignment. Please indicate below how you would like to team up with the classmates for the term project. The recommended team size is 3 (people/team).

SQL Commands



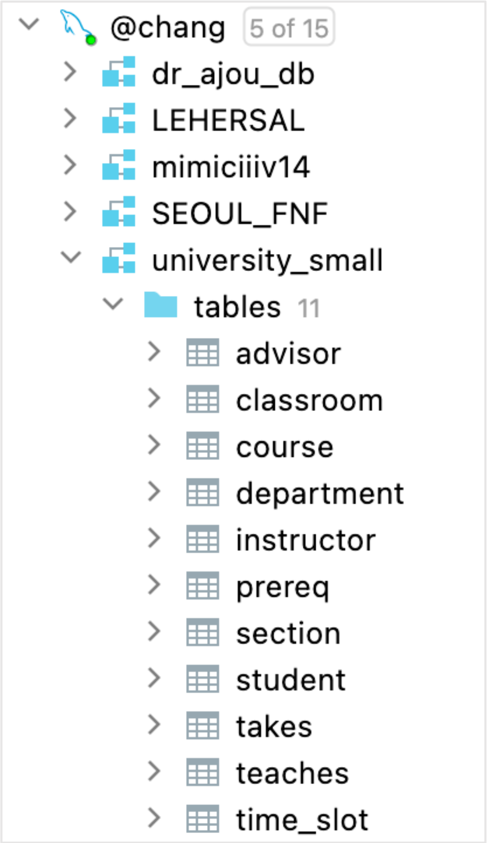
Data Definition Language

- 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

- Three key commands
  - CREATE
  - ALTER
  - DROP

CREATE DATABASE

- To initialize a new database
- Basic syntax:
  CREATE DATABASE *database_name*
  - One can specify the default character encoding method along with this command
    - **CREATE DATABASE** test
      DEFAULT CHARACTER SET utf8 utf8mb4
      COLLATE utf8_unicode_ci;
      - Collation: a set of rules that defines how to compare and sort character strings
  - After creating a database, to use it
    USE *database_name*

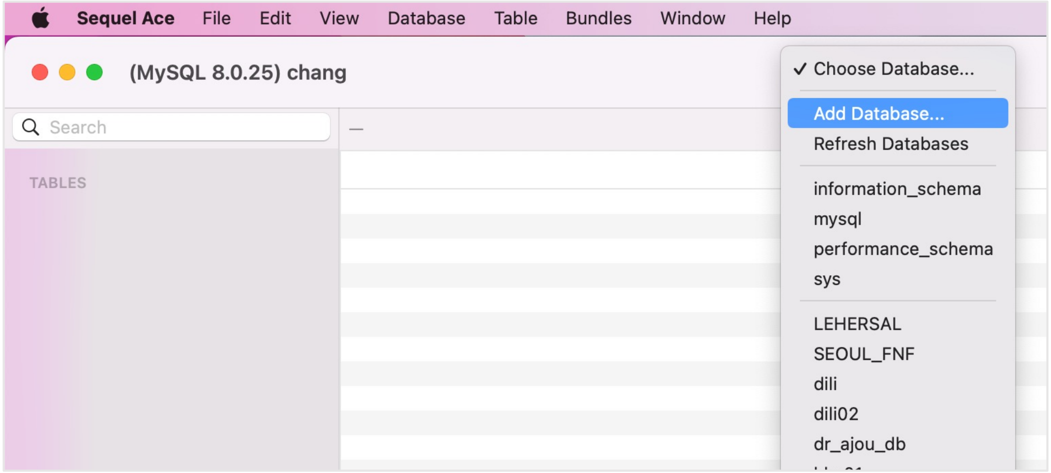


@chang 5 of 15
- dr_ajou_db
- LEHERSAL
- mimiciiiv14
- SEOUL_FNF
- university_small
  - tables 11
    - advisor
    - classroom
    - course
    - department
    - instructor
    - prereq
    - section
    - student
    - takes
    - teaches
    - time_slot

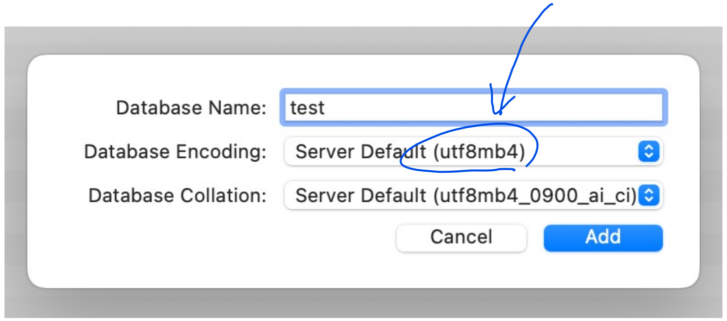
- Ref: https://dev.mysql.com/doc/refman/8.0/en/charset-charsets.html

Example: Creating a Database on Sequel Pro

- Creating a new database

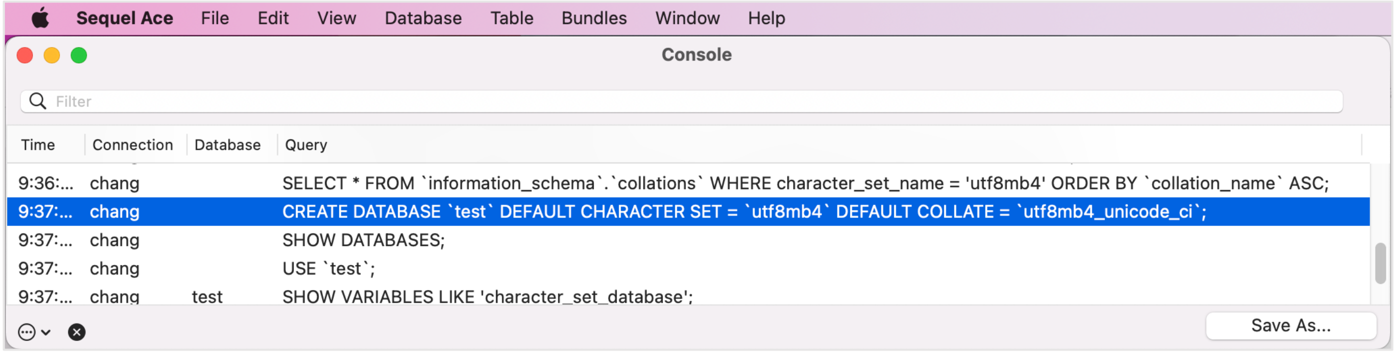


이모티콘도 표현가능.


Database Name: test

Database Encoding: Server Default (utf8mb4)

Database Collation: Server Default (utf8mb4_0900_ai_ci)



```
Time      Connection  Database  Query
9:36:...  chang                 SELECT * FROM `information_schema`.`collations` WHERE character_set_name = 'utf8mb4' ORDER BY `collation_name` ASC;
9:37:...  chang                 CREATE DATABASE `test` DEFAULT CHARACTER SET = `utf8mb4` DEFAULT COLLATE = `utf8mb4_unicode_ci`;
9:37:...  chang                 SHOW DATABASES;
9:37:...  chang                 USE `test`;
9:37:...  chang       test      SHOW VARIABLES LIKE 'character_set_database';
```

Agenda

- SQL DDL (Data Definition Language)

CREATE TABLE

- To create a new table
- Basic syntax:

```
CREATE TABLE table_name(
    Col1_name        data_type[(size)],
    Col2_name        data_type[(size)]
)
```

- E.g., Creating a table with four columns
  - CREATE TABLE books(

```
ISBN                CHAR(20),
Title               CHAR(50),
AuthorID            INTEGER,
Price               FLOAT)
```

Data Types in SQL

- Following categories of data types exist in most DBMSs
  - String data
  - Numeric data
  - Temporal data
  - Large objects

String Data 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*

    it just sepcify the maximum length. => n may not be the same as assigned storage.
    - 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

off records:

  - **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

these data type is not stored in a table. but in the external storage. So it links to the actual space in the external storage. which means it takes more time to retrieve

String Data 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

Numeric Data in SQL

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

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

Numeric Data 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

Numeric Data 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

Numeric Data 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

even though you want 3.30 but
in floating digits format
it actually represent that number.

Temporal Data 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)

Temporal Data in SQL

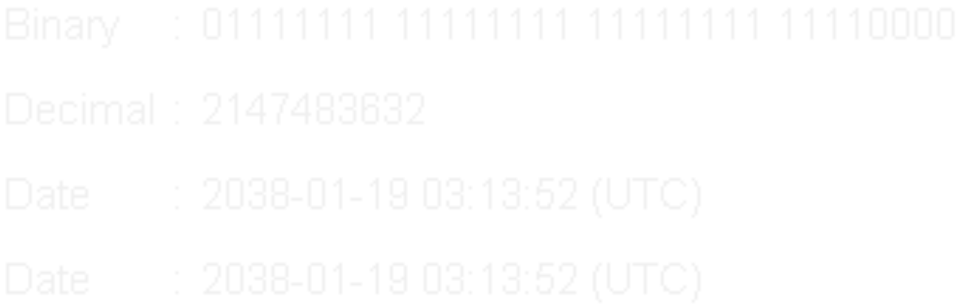
- SQL Data Types
  - **TIMESTAMP(*n*)**: Unix time (time since Jan 1, 1970)
    - A way to track time as a running total of seconds
    - Range: 1970-01-01 00:00:01 UTC to 2038-01-19 03:14:07 UTC
    - Typically used for logging (keeping records of all the system events)
    - *URL: https://time.is/Unix*

UNIX
TIME

SINCE 1970

Temporal Data in SQL

- SQL Data Types
  - **TIMESTAMP(*n*)**: Unix time (time since Jan 1, 1970)
    - A way to track time as a running total of seconds
    - 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)
    - *URL: https://time.is/Unix*



```
Binary  : 01111111 11111111 11111111 11110000
Decimal : 2147483632
Date    : 2038-01-19 03:13:52 (UTC)
Date    : 2038-01-19 03:13:52 (UTC)
```

Temporal Data 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

Large Objects 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]

On-record

off-record

  - **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*

  $(A_1\ D_1, A_2\ D_2, \ldots, A_n\ D_n,$

  $(\mathit{integrity\text{-}constraint}_1),$

  $\ldots,$

  $(\mathit{integrity\text{-}constraint}_k))$

  - *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** ($A_1, \ldots, A_n$)
  - Foreign key: **FOREIGN KEY** ($A_\mathrm{m}, \ldots, A_n$) **REFERENCES** *r*
  - Unique key: **UNIQUE** ($A_1, \ldots, A_n$)
  - Not null: **NOT NULL**
  - Value constraints: **CHECK** (*constraint*), **DEFAULT**
- 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 KEY and UNIQUE Constraints

- 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

Examples

```
• 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);
```

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 (inserted with the default value):
  - INSERT INTO neighbors (name)
    VALUES (‘Elmo’);

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

More Examples

- CREATE TABLE neighbors(

```
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

Integrity Constraints Recap

- Primary key, foreign key, and unique (candidate key) can be specified with DDL
  - A single or multiple columns can be specified as a key
  - Once a set of columns have been declared unique, any duplicate inputs are rejected

```
CREATE TABLE studio (
        ID              NUMERIC(5,0),
        name            VARCHAR(20),
        city            VARCHAR(20),
        state           CHAR(2),
);
```

Integrity Constraints Recap

- Primary key, foreign key, and unique (candidate key) can be specified with DDL
  - A single or multiple columns can be specified as a key
  - Once a set of columns have been declared unique, any duplicate inputs are rejected

```
CREATE TABLE studio (
        ID              NUMERIC(5,0),
        name            VARCHAR(20),
        city            VARCHAR(20),
        state           CHAR(2),
        UNIQUE(name)
);
```

Integrity Constraints Recap

- Primary key, foreign key, and unique (candidate key) can be specified with DDL
  - A single or multiple columns can be specified as a key
  - Once a set of columns have been declared unique, any duplicate inputs are rejected

```
CREATE TABLE studio (
        ID              NUMERIC(5,0),
        name            VARCHAR(20),
        city            VARCHAR(20),
        state           CHAR(2),
        UNIQUE(name),
        UNIQUE(city, state),
);
```

Integrity Constraints Recap

- Primary key, foreign key, and unique (candidate key) can be specified with DDL
  - A single or multiple columns can be specified as a key
  - Once a set of columns have been declared unique, any duplicate inputs are rejected

```
CREATE TABLE studio (
        ID                  NUMERIC(5,0),
        name                VARCHAR(20),
        city                VARCHAR(20),
        state               CHAR(2),
        PRIMARY KEY(ID),
        UNIQUE(name),
        UNIQUE(city, state),
);
```

Integrity Constraints Recap

- Primary key, foreign key, and unique (candidate key) can be specified with DDL
  - A single or multiple columns can be specified as a key
  - Once a set of columns have been declared unique, any duplicate inputs are rejected

```
CREATE TABLE studio (
        ID              NUMERIC(5,0) PRIMARY KEY,
        name            VARCHAR(20) UNIQUE,
        city            VARCHAR(20),
        state           CHAR(2),
        UNIQUE(city, state),
);
```

Integrity Constraints Recap

- Primary key, foreign key, and unique (candidate key) can be specified with DDL
  - A single or multiple columns can be specified as a key
  - Once a set of columns have been declared unique, any duplicate inputs are rejected

```
CREATE TABLE studio (
        ID              NUMERIC(5,0) PRIMARY KEY,
        name            VARCHAR(20) UNIQUE,
        city            VARCHAR(20),
        state           CHAR(2),
        UNIQUE(city, state),
         FOREIGN KEY (state) REFERENCES states
);
```

Integrity Constraints Recap

- **NOT NULL** – disallowing null values
  - Null values indicate that the data is not known
  - These can cause problems in querying database
  - The Primary Key columns automatically prevent null being entered
  - *C.f.*, **NULL** – can be used to explicitly allow null values

```
CREATE TABLE studio (
        ID              NUMERIC(5,0) PRIMARY KEY,
        name            VARCHAR(20) NOT NULL,
        city            VARCHAR(20) NULL,
        state           CHAR(2) NOT NULL
);
```

Integrity Constraints Recap

- **DEFAULT** – A default value can be inserted in any column with this keyword
  - *E.g.,*

```
CREATE TABLE movies(
        movie_title         VARCHAR(40) NOT NULL,
        release_date        DATE  DEFAULT sysdate  NULL,
        genre               VARCHAR(20) DEFAULT ‘Comedy’
                            CHECK genre IN (‘Comedy’, ‘Action’, ‘Drama’)
)
```

  - In MySQL,
    -

```
CREATE TABLE movies(
        movie_title         VARCHAR(40) NOT NULL,
        release_date        DATE  DEFAULT CURRENT_TIMESTAMP  NULL,
        genre               VARCHAR(20) DEFAULT ‘Comedy’
                            CHECK genre IN (‘Comedy’, ‘Action’, ‘Drama’)
)
```

Integrity Constraints Recap

- **CHECK** – Allows the inserted value to be checked
  - *E.g.*,

```
CREATE TABLE movies(
            movie_title       VARCHAR(40) PRIMARY KEY,
            release_date      DATE,
            budget            INTEGER CHECK (budget > 50000)
    )
```

  - Table-level constraints can be defined; *E.g.*,

```
CREATE TABLE movies(
            movie_title       VARCHAR(40) PRIMARY KEY,
            release_date      DATE,
            budget            INTEGER CHECK (budget > 50000),
            CONSTRAINT release_date_const
            CHECK (release_date BETWEEN '01-Jan-2000' AND '31-Dec-2009')
    )
```

Table Updates (Updating Table Schemas)

- **DROP**: Used to remove elements from a database, such as tables
  - **DROP TABLE** *r*
    - Remove relation *r*
    - *C.f.*, **TRUNCATE** (**TABLE**) *r* is used to delete the data inside a table, but not the table itself

- **ALTER**: Used to make changes to the table schema
  - **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 **MODIFY** remark CHAR(20);
  - **ALTER TABLE** time_slot_backup **DROP** remark;

Table Updates (Updating Table Schemas)

- Examples
  - Drop a column
    - **ALTER TABLE** salary
      DROP COLUMN instructor;
  - DROP a PRIMARY KEY Constraint
    - **ALTER TABLE** instructor
      DROP PRIMARY KEY;
  - DROP a FOREIGN KEY Constraint
    - **ALTER TABLE** instructor
      DROP FOREIGN KEY instructor_ibfk_1;
  - DROP DEFAULT
    - **ALTER TABLE** student
      ALTER tot_cred **DROP DEFAULT**;

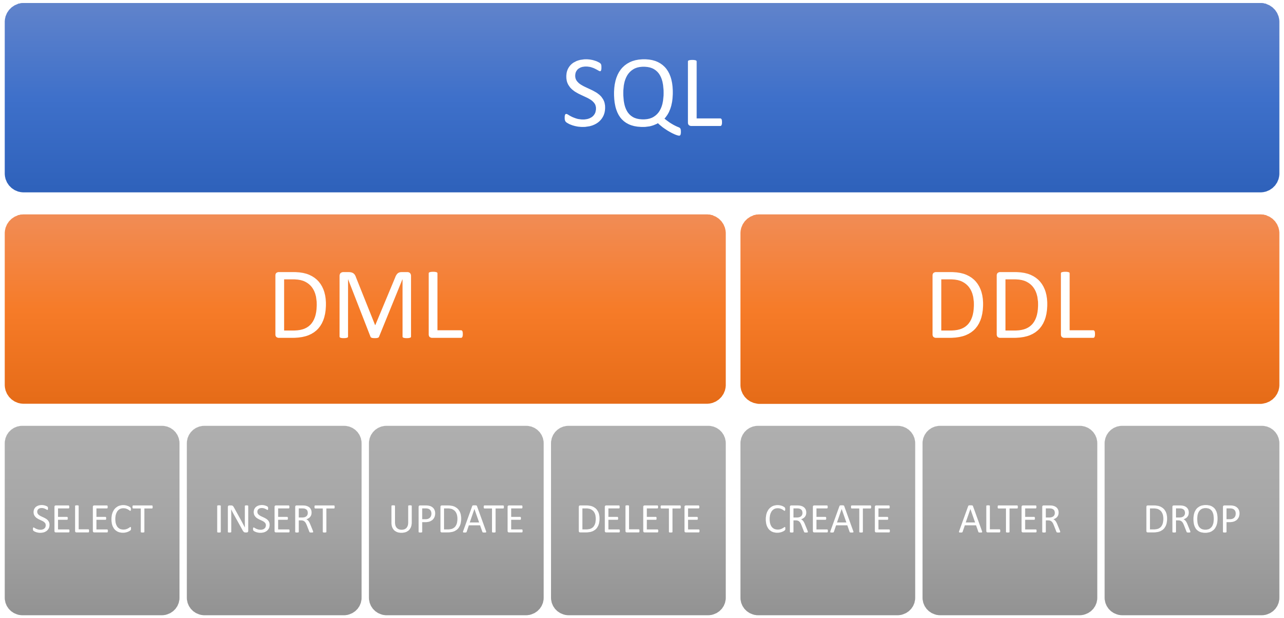
Table Updates (Updating Table Schemas)

- Examples
  - Drop a database
    - **DROP DATABASE** university;

Table Updates (Updating Tuples)

- INSERT
  - **INSERT INTO** *instructor*
    VALUES ('10211', 'Smith', 'Biology', 66000)
- DELETE
  - **DELETE FROM** *student*
    - Remove all tuples from the *student* relation
    - **TRUNCATE TABLE** *student*

SQL Commands



SQL

DML

DDL

SELECT

INSERT

UPDATE

DELETE

CREATE

ALTER

DROP

INSERT

- Add a new tuple to *course*
  - **INSERT INTO** *course*
VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
- or equivalently
  - **INSERT INTO** *course* (*course_id*, *title*, *dept_name*, *credits*)
VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
- Add a new tuple to *student* with *tot_creds* set to null
  - **INSERT INTO** *student*
VALUES ('3003', 'Green', 'Finance', *null*);

INSERT

- Inserting results of other SELECT query
  - Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000
    - **INSERT INTO** *instructor*
      SELECT *ID, name, dept_name, 18000*
      FROM *student*
      WHERE *dept_name* = 'Music' **AND** *total_cred* > 144;
  - The **SELECT FROM WHERE** statement is evaluated fully before any of its results are inserted into the relation
    - Otherwise queries like
      INSERT INTO *table*1 **SELECT** * **FROM** *table*1
      would cause problem

UPDATE

- Basic syntax
  - Updating a table
    - **UPDATE** *tablename*
      SET *col1_name = new_col1_value, col2_name = new_col2_value, …*;
  - Updating a table with conditions
    - **UPDATE** *tablename*
      SET *col1_name = new_col1_value, col2_name = new_col2_value, …*
      WHERE *predicate*;

UPDATE

- Give a 5% salary raise to all instructors
  - **UPDATE** *instructor*
    SET *salary* = *salary* * 1.05

- Give a 5% salary raise to those instructors who earn less than 70000
  - **UPDATE** *instructor*
    SET *salary* = *salary* * 1.05
    WHERE *salary* < 70000;

- Give a 5% salary raise to instructors whose salary is less than average
  - **UPDATE** *instructor*
    SET *salary* = *salary* * 1.05
    WHERE *salary* < (**SELECT AVG**(salary) **FROM** *instructor*);

UPDATE

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two UPDATE statements:

```
UPDATE instructor
SET salary = salary * 1.03
WHERE salary > 100000;

UPDATE instructor
SET salary = salary * 1.05
WHERE salary <= 100000;
```

  - The order is important
  - Can be done better using the **case** statement (next slide)

CASE Statement for Conditional Update

- The following query is equivalent to the previous UPDATE queries

  -
    ```
    UPDATE instructor
    SET salary = CASE
                        WHEN salary <= 100000 THEN salary * 1.05
                        ELSE salary * 1.03
                     END
    ```

UPDATE with Scalar Subqueries

- Recompute and update *tot_creds* value for all students
  - **UPDATE** *student S*
    SET *tot_cred* = (**SELECT SUM**(*credits*)
    FROM *takes, course*
    WHERE *takes.course_id* = *course.course_id* **AND**
    S.ID= *takes.ID* **AND**
    takes.grade <> 'F' **AND**
    takes.grade **IS NOT NULL**);

DELETE

- Basic syntax
  - To remove specific rows
    - **DELETE FROM** *tablename*
      WHERE *predicate*;
  - To remove all rows
    - **DELETE FROM** *tablename*;
    - This is equivalent to **TRUNCATE**:
      TRUNCATE (**TABLE**) *tablename*;
    - One cannot truncate a table with foreign key constraints
      - Must disable the constraints first (we will cover **ALTER** when we study SQL DDL):
        ALTER TABLE *tablename*
        DISABLE CONSTRAINT *constraint_name*;

DELETE

- Delete all instructors

  -
    ```
    DELETE FROM instructor;
    ```

- Delete all instructors from the Finance department

  -
    ```
    DELETE FROM instructor
    WHERE dept_name= 'Finance';
    ```

- Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building

  -
    ```
    DELETE FROM instructor
    WHERE dept name IN (SELECT dept name
                        FROM department
                        WHERE building = 'Watson');
    ```

DELETE

- Delete all instructors whose salary is less than the average salary of instructors
  - Example:

```
DELETE FROM instructor
 WHERE salary < (SELECT AVG (salary)
                 FROM instructor);
```

- Issue: as we delete tuples from *instructor*, the average salary changes
  - Solution used in SQL:
    1. First, compute **AVG**(*salary*) and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing **AVG** or retesting the tuples)

EOF

- Coming next:
  - Designing a database