ECE30030/ITP30010 Database Systems

SQL DDL

Reading: Chapter 3

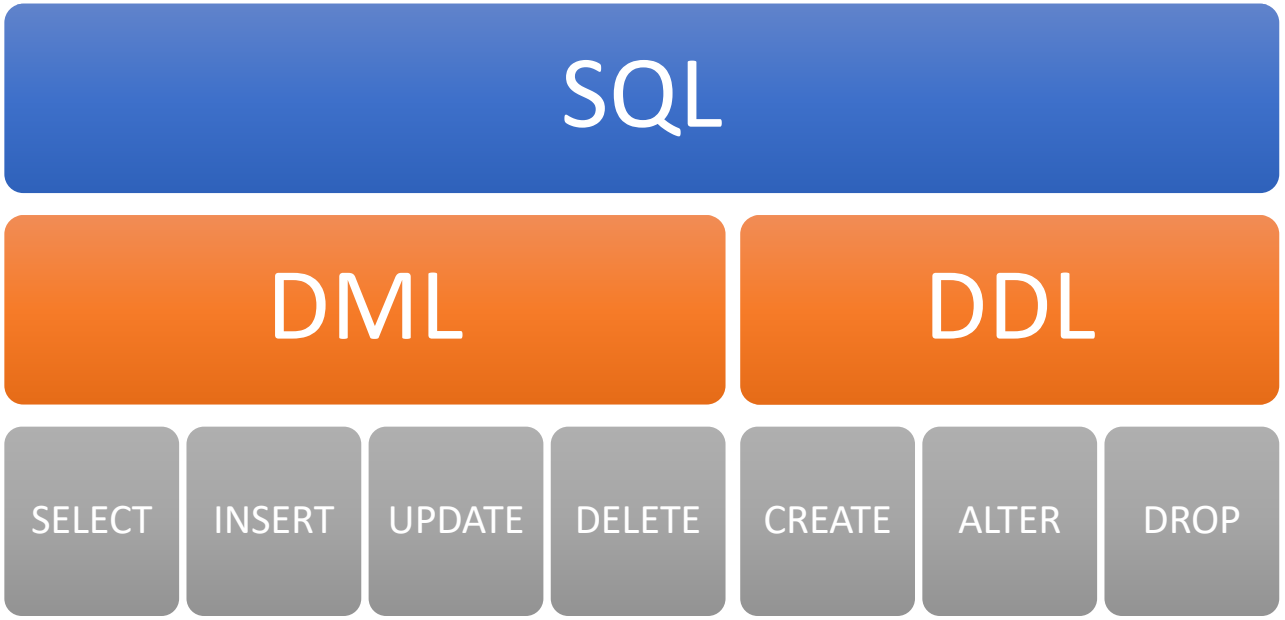
Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

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

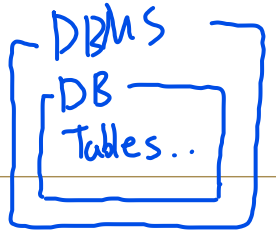
Table structure.

Things you can do with DDL.

an additional data structure along with the table (associate) to increasing a performance in reading data.

- Three key commands
  - CREATE
  - ALTER
  - DROP

CREATE DATABASE

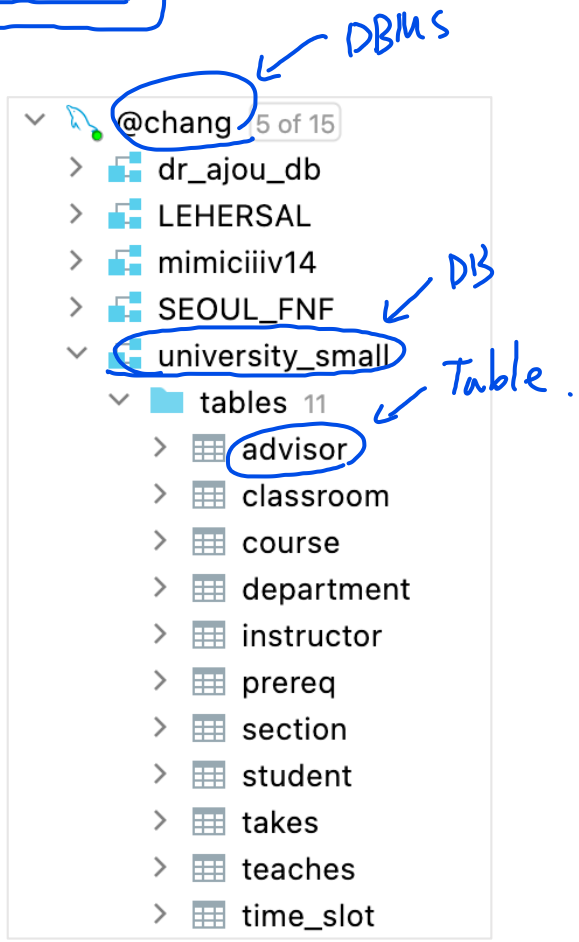


- To initialize a new database
- Basic syntax:

  CREATE DATABASE *database_name*

  - One can specify the default character encoding method along with this command
    - **CREATE DATABASE** test
      DEFAULT CHARACTER SET utf8
      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

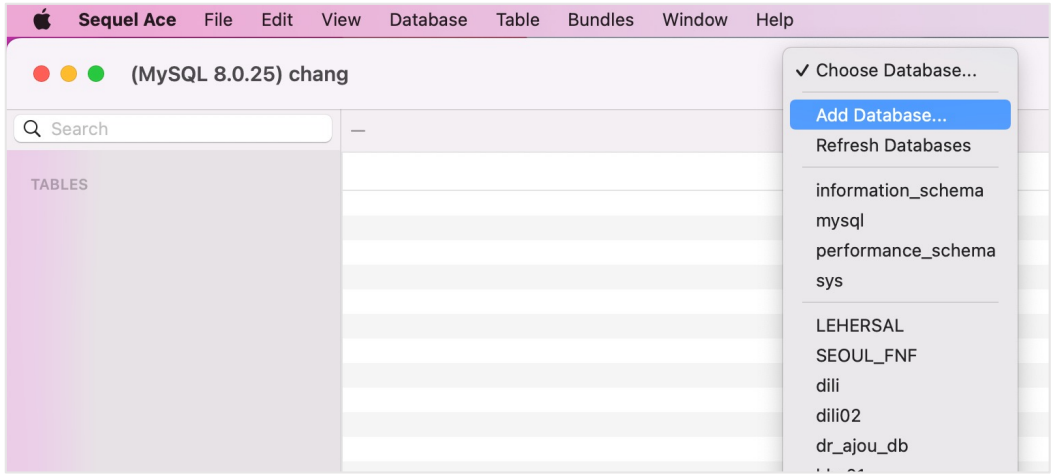
    USE *database_name*

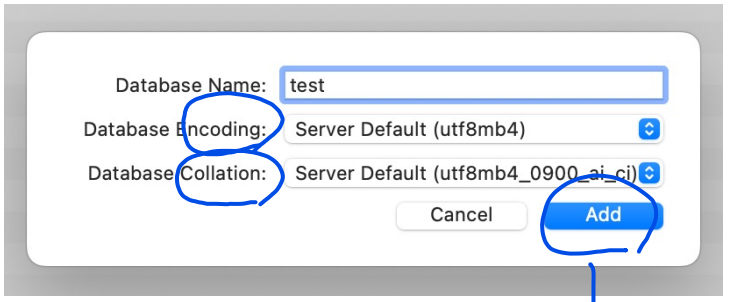


- Ref: https://dev.mysql.com/doc/refman/8.0/en/charset-charsets.html

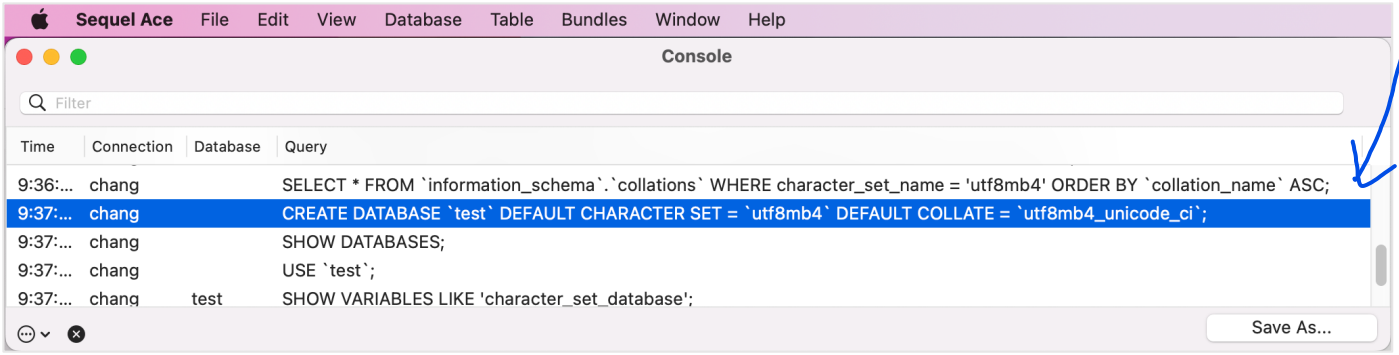
Example: Creating a Database on Sequel Pro

- Creating a new database





Automatically make query for it.



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*
    - 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

allocate a fixed size of space. regardless of data type.

allocate the storage according to the size of data value.

16MB. 4GB. ← stored in storage not in a tree.

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

for null char. to indicate the end of the string.

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

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*

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]
  - **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, …, A_n D_n,
  (*integrity-constraint*$_1$),
  …,
  (*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** (A_1, ..., A_n)
  - Not null: **NOT NULL**
  - Value constraints: **CHECK** (*constraint*), **DEFAULT**

identifier of records in the table.
if you have pk value.
you can specify the corresponding record.

- 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);
```

in department table, there should be dept_name attribute.

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

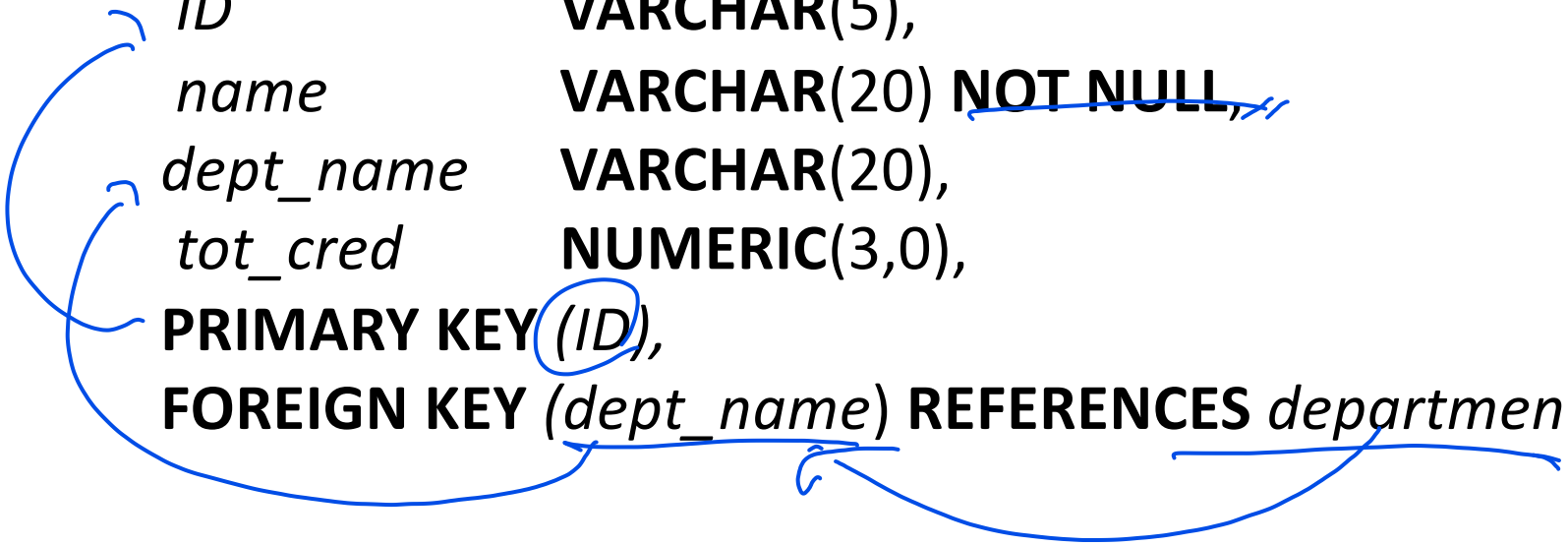
Can determine a record uniquely but is still not a PK

PK just can be a combination of multiple attributes.

To read faster.

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*);

Simpler version.

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*);

a combination of attributes as a PK value.

foreign key can be also a combination of attributes.

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*);

If It is not set then follow. Default value.

But what if user try to insert Null value?

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*(
  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),
);
```

Can have multiple unique.. and declared multiple times.

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 (state)
);
```

or states.state.

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
);
```

It can explicitly say it can have null value but not necessary.

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')

)

sql standard meaning current time.

If you insert a new record with null value for release_date then DBMS will automatically put sysdate for that.

then release_date will never have null value? → No → You can update it to Null later.

MySQL doesn't have sysdate but this or NOW()

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')
)
```

check..

Constraint name. but why? → to be easy to maintain. easier to find. or comments.

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*

Remove all records But not a table structure.

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~~ instructor
      DROP COLUMN ~~instructor~~ salary;
  - 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**;

it does not mean it drops the records of primary key but just like its primary key constraint up.

Table Updates (Updating Table Schemas)

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

EOF

- Coming next:
  - Designing a database