



Xi'an Jiaotong-Liverpool University

西交利物浦大學

SQL - Tables and Data Part 1

Jianjun Chen

Contents

- SQL background
 - Database Containment Hierarchy
 - introduction to SQL software
- Creating Tables
 - Syntax
 - Data types.
 - Column options
- Tuple Manipulation
 - Insert, update, delete
- Constraints.
 - Primary key,
 - Unique key
 - Domain

SQL Background Info

Database Containment Hierarchy, introduction to SQL software

About SQL

- SQL stands for “Structured Query Language”.
- SQL consists of two parts:
 - Data definition language (DDL).
 - Data manipulation language (DML).
- Today, we will cover the DDL and a small portion of DML:
 - Create tables and their constraints.
 - Changing columns/constraints of tables.
 - Adding/Updating/Removing tuples.

SQL Format in the Slides

- SQL statements will be written in
 - **BOLD COURIER FONT**
- SQL keywords are not case-sensitive, but my slides will present SQL keywords in uppercase for clarity.
- Table names, column names etc. can be configured to be case sensitive. For example:

SELECT (sName) **FROM** Student;

- SQL statements should end with semi-colons.
- Two dashes followed by a space will comment out that line.

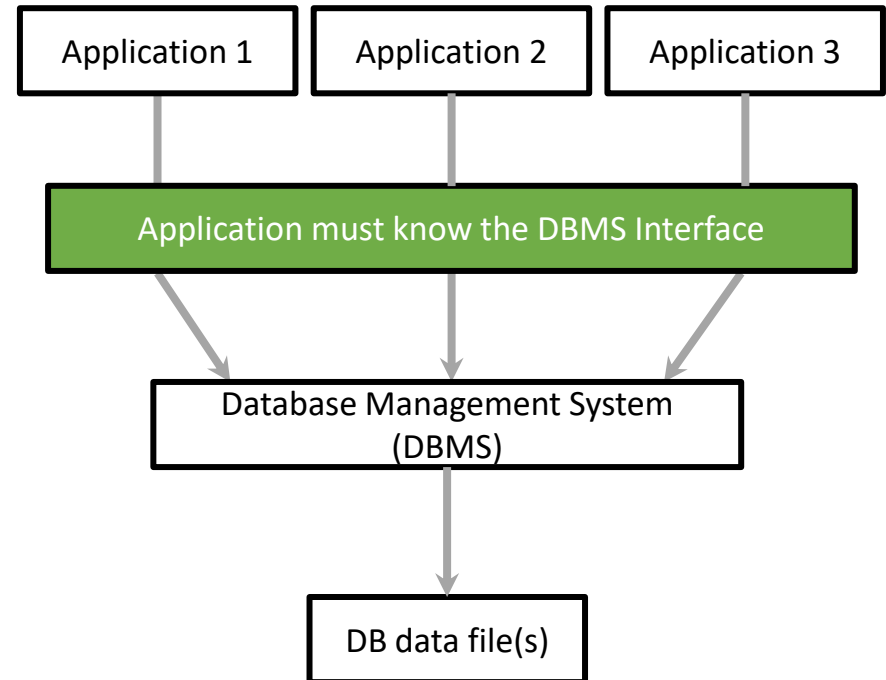
```
-- commented out  
--not correctly commented out
```

Database Containment Hierarchy

- A computer may have one or more **clusters**.
 - A cluster is a database server.
 - = a computer can [run multiple database servers](#).
- Each cluster contains one or more **catalogs**.
 - Catalog is just another name for “database”.
- Each catalog consists of set of **schemas**.
 - Schema is a namespace of tables, and security boundary.
 - A good discussion about this is available [here](#).
- A schema consists of tables, views, domains, assertions, collations, translations, and character sets. All have same owners.

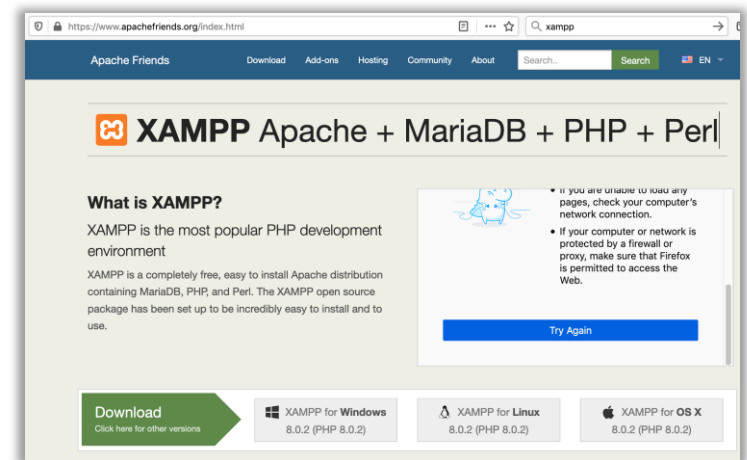
SQL Database and Related Software

- DBMS is an application that usually runs on a server.
 - MySQL, MariaDB, PostgreSQL, Oracle...
- Different client applications can access this server simultaneously.
 - DBeaver, MySQL workbench, PhpMyAdmin
 - Your own programs.
- This module:
 - MySQL/MariaDB as DB server.
 - PhpMyAdmin as client for labs.



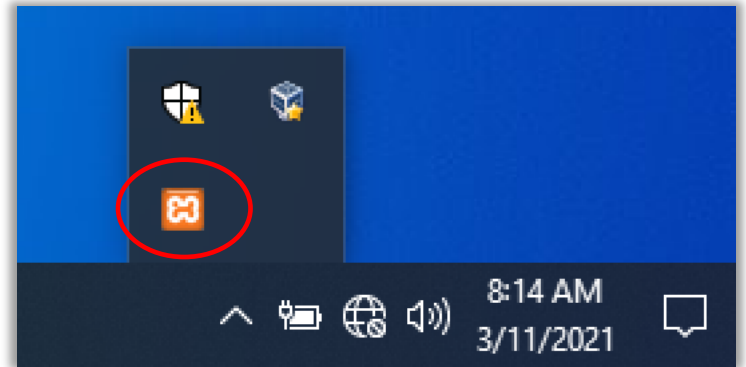
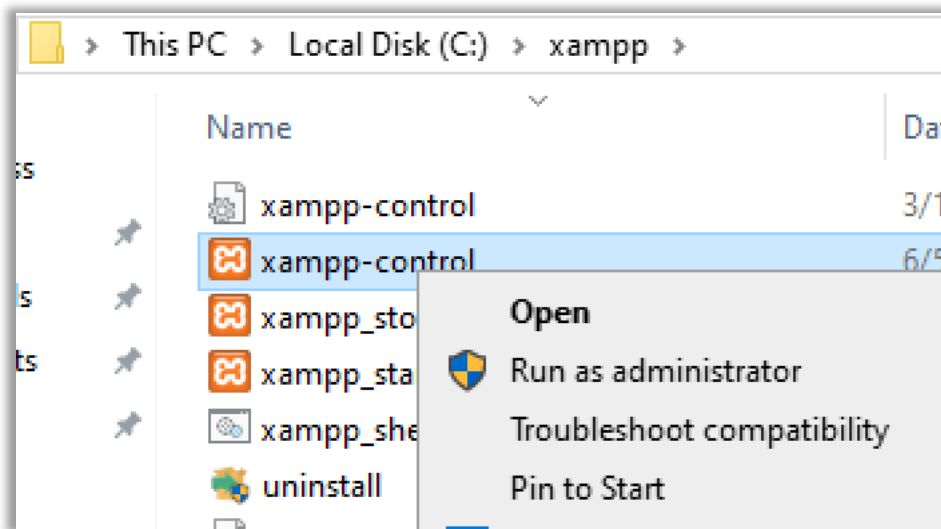
XAMPP

- MySQL and PhpMyAdmin are only accessible on campus.
 - It is recommended to install XAMPP on your own computer if you want the same environment. (Off-campus students should install it)
- “XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl.”
- I will upload the installer program to the box and the link will be available on the LearningMall.
- You may also download from [here](#).
 - Connection can be slow though



Setting up XAMPP (Windows)

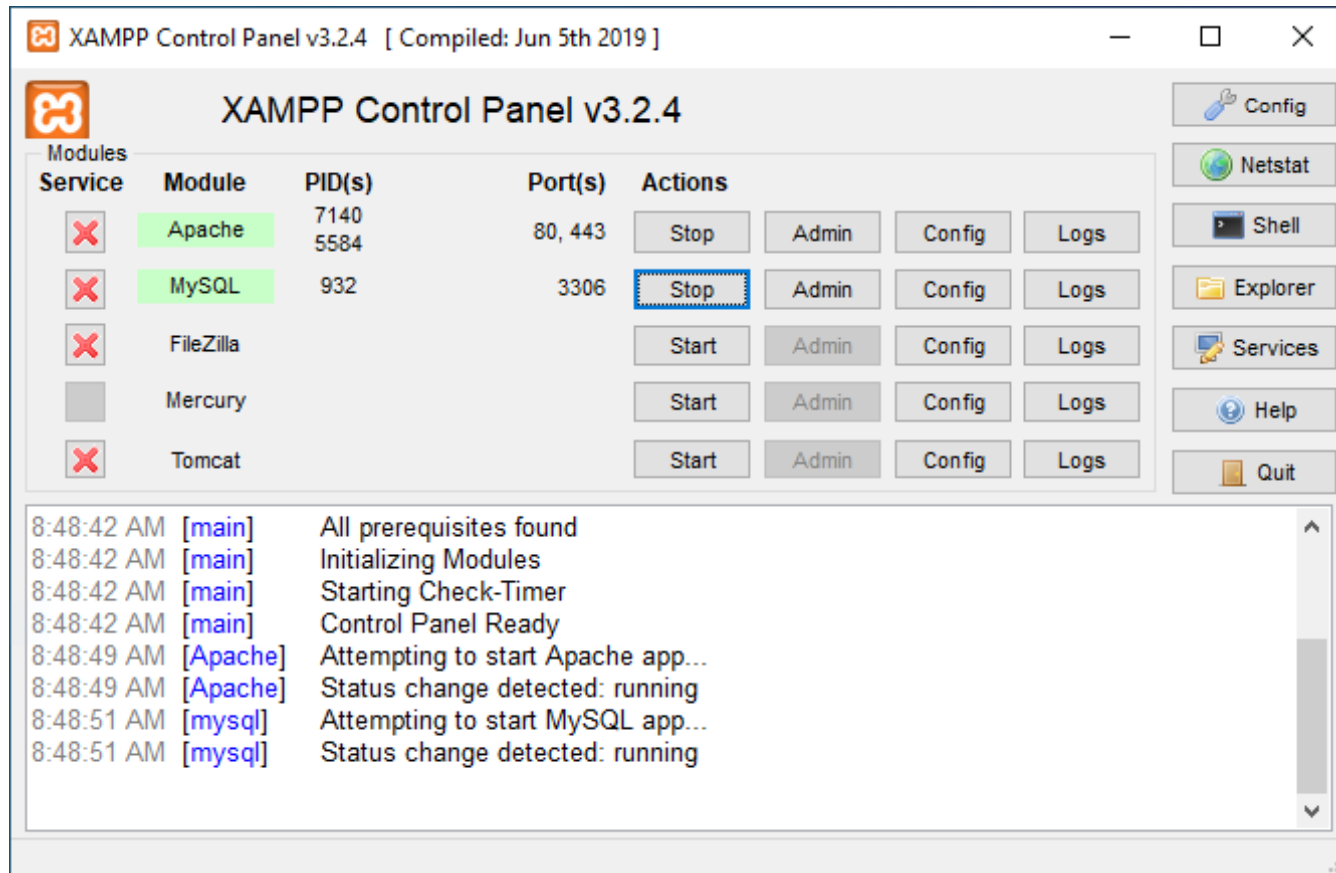
- Install
- Always run “xampp-control.exe” as administrator.



- You can start multiple instances of xampp-control, but don't do that!
 - Check the system tray before starting xampp. If there's one running already, just use that one.

Starting Database (Windows)

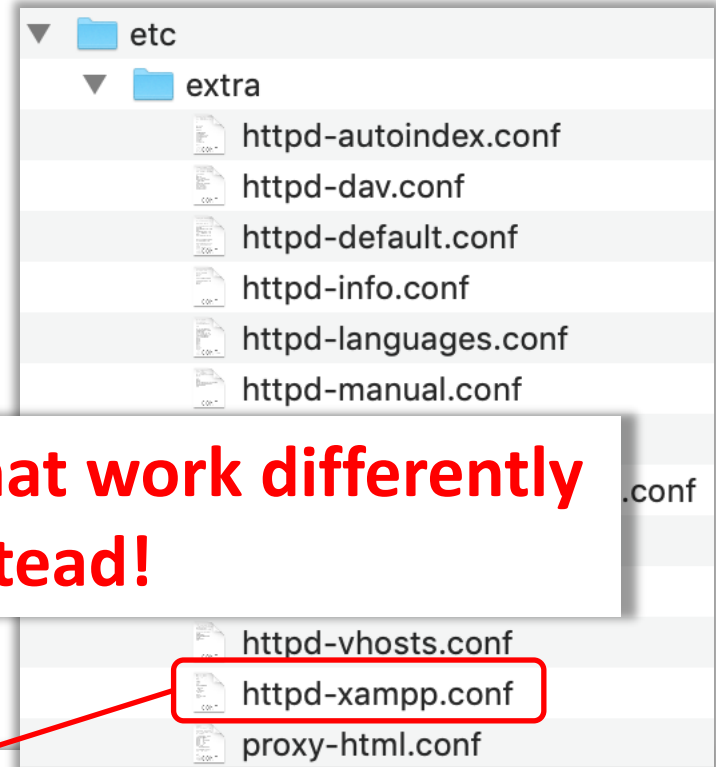
- Start both Apache and PhpMyAdmin



Setting up XAMPP (Mac)

- Install
- Change virtual machine access rights.

**XAMPP on Mac has 2 versions that work differently
Please follow the live lecture instead!**

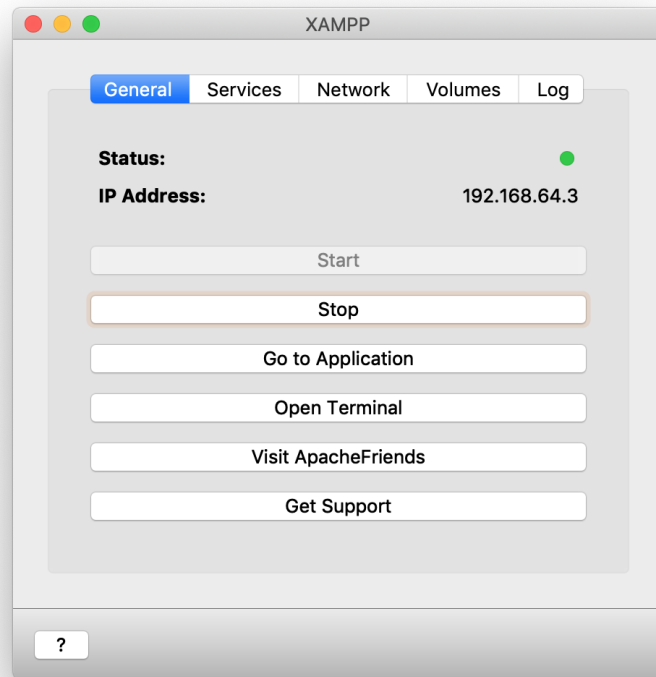


```
# since XAMPP 1.4.3
<Directory "/opt/lampp/phpmyadmin">
    AllowOverride AuthConfig Limit
#    Require local
    Require all granted
    ErrorDocument 403 /error/XAMPP_FORBIDDEN.html.var
</Directory>
```

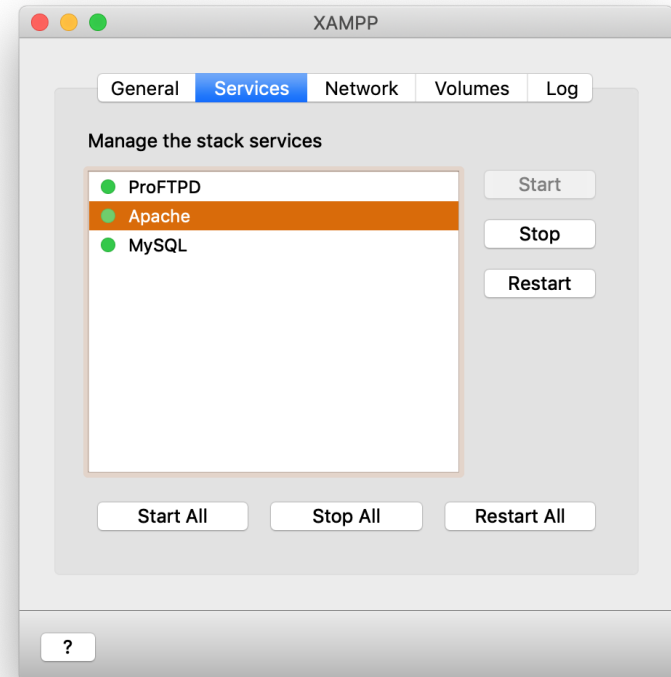
Use this line

Starting Database (Mac)

Starting the virtual machine



Start both Apache and MySQL



Using PhpMyAdmin

- Explained in my lecture.

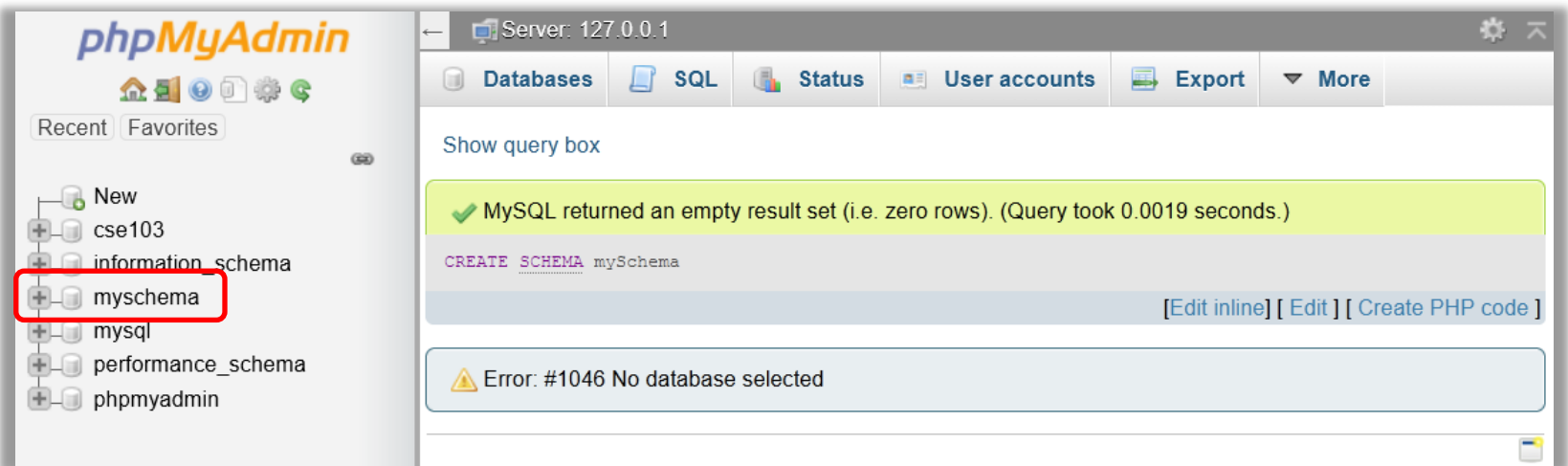
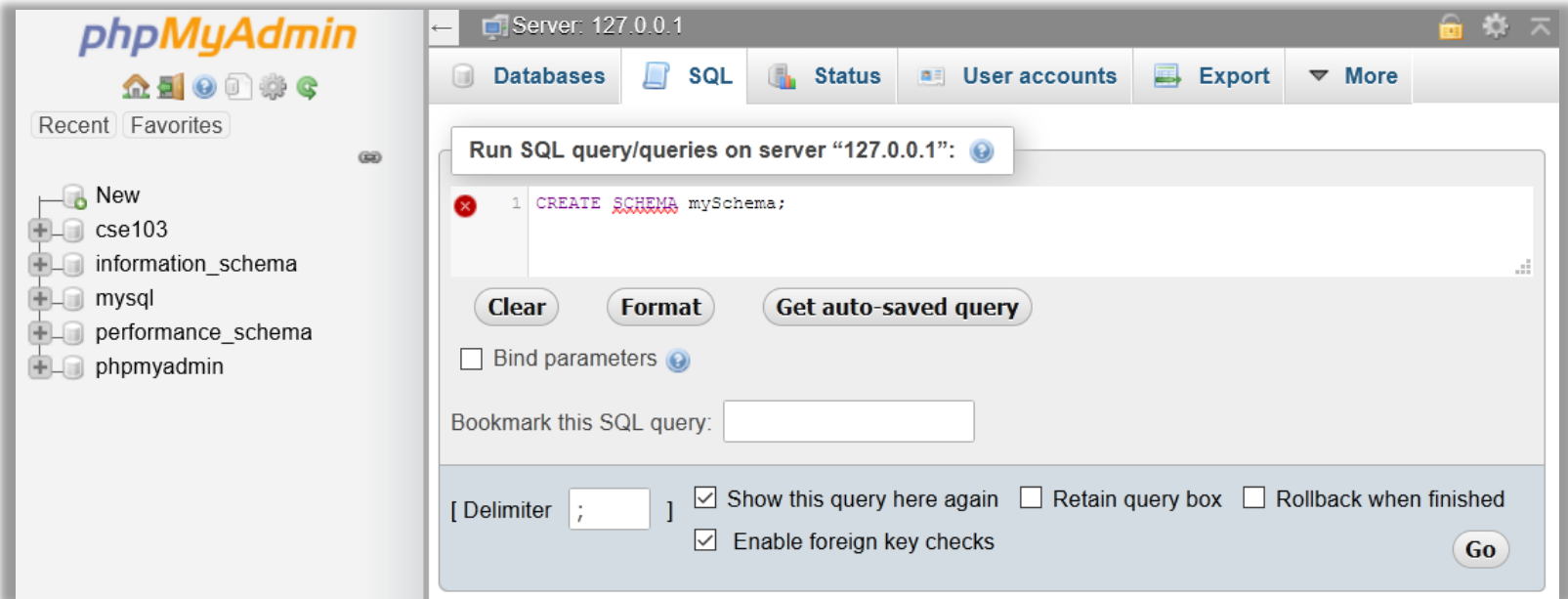
Creating a Database

- First, we need to create a schema

- **CREATE SCHEMA** name ;
 - **CREATE DATABASE** name ;

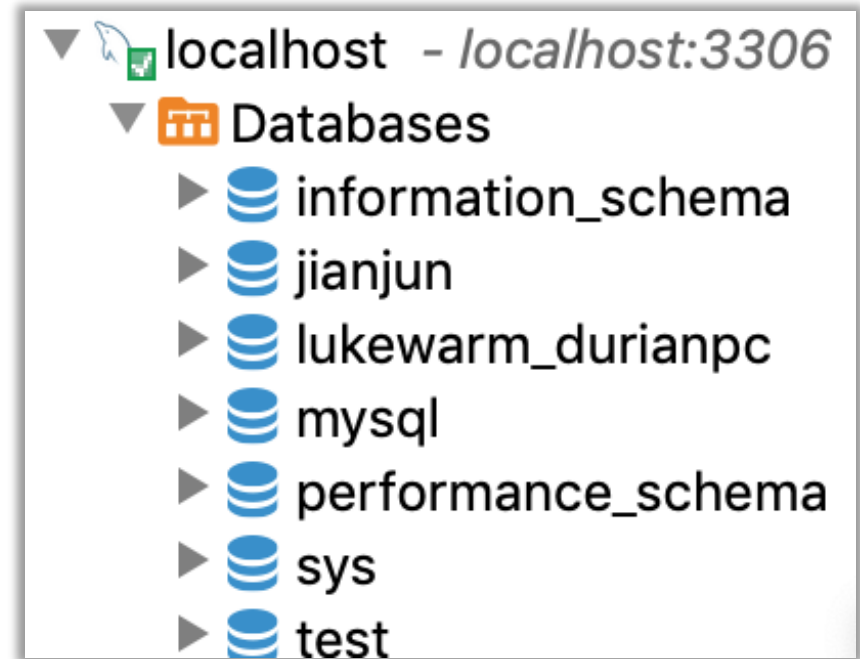
} Same

- If you want to create tables in this schema, you need to tell MySQL to “enter” into this schema, type:
 - **USE** name ;
- Then all following statements like “SELECT” or “CREATE TABLE”, in the same script, will be executed in this schema.
 - Just like a file manager, after entering a folder, all operations will apply to this folder.



Some Notes

- Avoid creating or modifying tables in these databases that are created automatically by the system:
 - information_schema
 - mysql
 - performance_schema
 - sys
- They are created for database administration purpose.



Error Messages

- Error messages from SQL server are not always informative.

Run SQL query/queries on database **test**:

```
1 create table `sometable`  
2 (  
3     column1 INT NOT NULL,  
4     column2 TEXT,  
5 );
```

Error


Static analysis:

1 errors were found during analysis.

1. A symbol name was expected! (near ")") at position 72)

SQL query: [Copy](#)

create table `sometable` (column1 INT NOT NULL, column2 TEXT,)

MySQL said: 

#1064 - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ')' at line 5

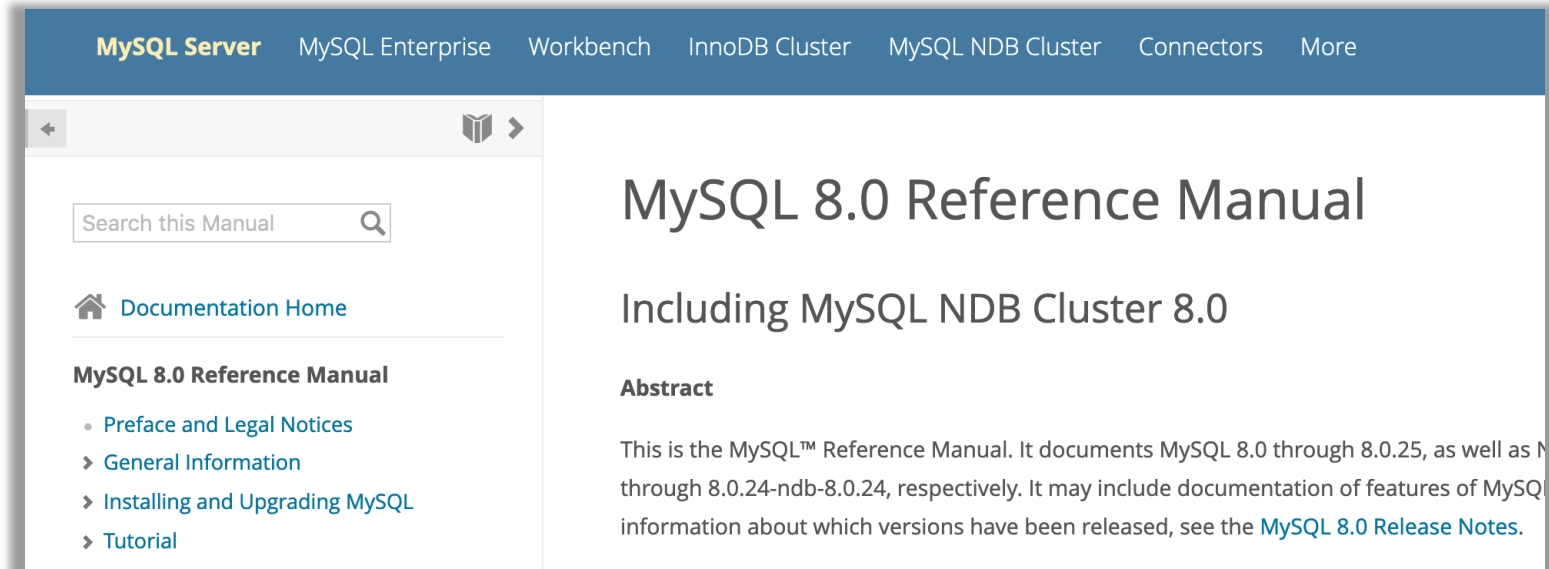
- As a result, remembering the correct syntax is very important when writing SQL statements.
 - Make sure you check the official manual for detailed syntax.
- Commenting out a part of the script and find out the source of the problem.

The Official Manual

- Being able to check the official manual is a very important skill.

<https://dev.mysql.com/doc/refman/8.0/en/>

- Looking up “how to xxxx” on Google/Baidu is not always a good idea.



SQL: Creating Tables


Syntax of “CREATE TABLE”

Data types in SQL

Syntax

To create tables in SQL, you need the “Create table” statement, the general syntax is shown below:

```
CREATE TABLE [IF NOT EXISTS] name (  
    col-name datatype [col-options],  
    :  
    col-name datatype [col-options],  
    [constraint-1],  
    :  
    [constraint-n]  
);
```



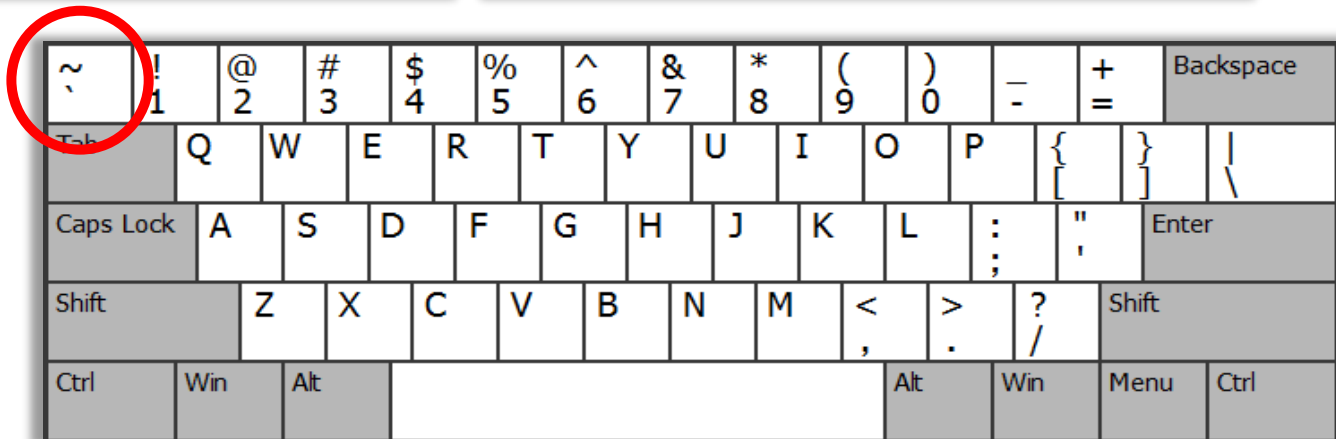
Contents between [] are optional

Table and Column Names

- In SQL, table or column names can have spaces, but it is **strongly not recommended**.
- If you insist to do so, you must enclose the name with a pair of (`) symbols.

```
CREATE TABLE mytable (  
    columnA INT,  
    columnB CHAR(11)  
);
```

```
CREATE TABLE `my table` (  
    `column A` INT,  
    `column B` CHAR(11)  
);
```



Numerical Data Types (Integers)

- **SMALLINT**

- Use 2 Bytes of memory
- range: -32768 to +32767

- **INT** or **INTEGER**

- Use 4 Bytes of memory
- range: -2147483648 to +2147483647
- A typical choice for integer.

- **BIGINT**

- Use 8 Bytes of memory.

```
CREATE TABLE `staff` (  
    `name` VARCHAR(12),  
    `staff_id` INT  
);
```

Numerical Data Types (Fixed Point)

- **DECIMAL [(M[,D])]** or **NUMERIC [(M[,D])]**

- Fixed point number.

- Example:

Things between [] are optional

```
CREATE TABLE `staff` (  
    `name` VARCHAR(12),  
    `staff_id` INT(11),  
    `salary` DECIMAL(5,2)  
);
```

Decimal(size,d)

12345666666.7777

M

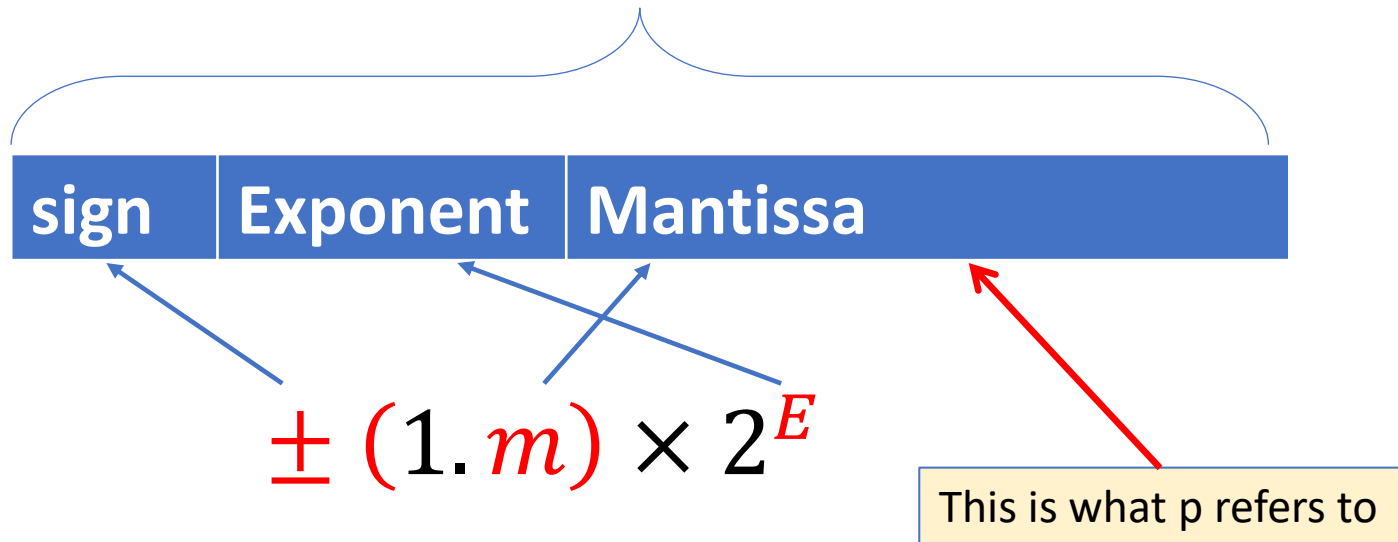
D

Numerical Data Types (Float)

- **FLOAT (p)**

- Floating point number in *IEEE 754 standard*
- represents the precision in bits. (Mantissa size)
- MySQL will automatically choose single precision or double precision based on the value of p.

Total number of bits is 32 for float and 64 for double



String Types

- **CHAR [(M)]**

- A fixed-length (**M** is the length, 0 ~ 255) string
- Always right-padded with spaces to the specified length when stored.
- E.g. if you store '**A**' to **CHAR (5)**, it will actually be '**A** ' inside the memory. But when you retrieve the value, the trailing spaces will be removed automatically (this behaviour can be turned on or off).

- **VARCHAR (M)**

- A variable-length string.
- The range of M is 0 to 65,535.

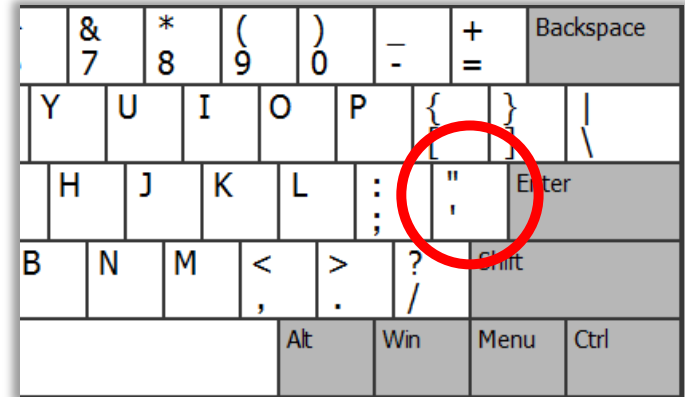
```
CREATE TABLE `staff` (  
    `name` VARCHAR(12),  
    `id_card` CHAR(6)  
);
```

- For MySQL:

- **CHAR** is faster, but occupies more memory
- **VARCHAR** is slower, but occupies less memory

String Values

- String values in SQL are surrounded by single quotes:
 - 'I AM A STRING'
- Single quotes within a string are doubled or escaped using \
 - 'I''M A STRING'
 - 'I\'M A STRING'
 - '' - is an empty string
- In MySQL, double quotes also work
 - (Not a standard)
- Example (after creating the table)



```
CREATE TABLE `staff` (  
    `name` VARCHAR(12),  
    `id_card` CHAR(6)  
);  
insert into `staff` values ('Daryl', 'STF001');
```

Date and Time

- **DATE**

- The supported range is '1000-01-01' to '9999-12-31'.
- MySQL displays DATE values in 'YYYY-MM-DD' format.

- **DATETIME** [(fsp)]

The number of digits is defined by fsp

- The supported range is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'
- Display format in MySQL: 'YYYY-MM-DD hh:mm:ss[.fraction]'

- Example:

```
CREATE TABLE `staff` (  
    `name` VARCHAR(12),  
    `id_card` CHAR(6)  
    `recruit_date` DATE,  
    `last_login` DATETIME  
);
```

Date and Time





- **TIMESTAMP**

- Similar to DATETIME, but stores the time as UTC time.
- When date time is looked up, the stored time will be converted to the time of the current timezone of the client.
- How to illustrate their difference? Enter the following table with data:





```
create table `ts_test` (  
    `time1` timestamp,  
    `time2` datetime  
);  
  
insert into `ts_test` values  
('2020-01-08 08:00:00', '2020-01-08 08:00:00');
```

Date and Time

- Assume your current timezone is UTC+8 (Beijing).
 - Look up the data in ts_test, you will find:

	 time1 	 time2 
1	2020-01-08 08:00:00	2020-01-08 08:00:00

- Now, in the system, change your timezone to UTC+0 (London)
 - Look up the data in ts_test again, you will now find:

	 time1 	 time2 
1	2020-01-08 00:00:00	2020-01-08 08:00:00

Column Options

```
col-name datatype [col-options]
```

- **NOT NULL:**

- values of this column cannot be null.

- **UNIQUE:**

- each value must be unique (similar to candidate key on a single attribute)

```
Example: age INT DEFAULT 12
```

- **DEFAULT** value:

- Default value for this column if not specified by the user.
- Does not work in MS Access.

Column Options

- **AUTO_INCREMENT** = baseValue:

```
CREATE TABLE Persons (  
    Id INT AUTO_INCREMENT,  
    ...  
)
```

You are not allowed to use
"AUTO_INCREMENT = 2"
inside CREATE TABLE

- Must be applied to a key column (primary key, unique key)
- a value (usually $\max(\text{col}) + 1$) is automatically inserted when data is added.
- You can also manually provide values to override this behaviour.
 - Add this line right after the CREATE TABLE statement:

```
ALTER TABLE Persons AUTO_INCREMENT = 100;
```

- Read the official manual [here](#).

Example

```
CREATE TABLE Persons (  
    id INT UNIQUE NOT NULL AUTO_INCREMENT,  
    lastname VARCHAR(255) NOT NULL,  
    firstname VARCHAR(255) ,  
    age INT DEFAULT 12,  
    city VARCHAR(255)  
) AUTO_INCREMENT = 5;
```


Implicit Default Values

- When a `DEFAULT` option is not used, the database may give implicit default values depending on the column data type:
 - If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause.
 - If the column cannot take `NULL` as a value, MySQL defines the column with no explicit `DEFAULT` clause.

```
CREATE TABLE t (  
    i INT NOT NULL  
);
```

```
INSERT INTO t VALUES ();  
INSERT INTO t VALUES (DEFAULT);  
INSERT INTO t VALUES (DEFAULT(i));
```

All statements above gives error in strict mode

- Please check the documentation for details:
<https://dev.mysql.com/doc/refman/8.0/en/data-type-defaults.html>.

Tuple Manipulation

INSERT, UPDATE, DELETE

INSERT

- INSERT add rows into the database:

```
INSERT INTO tablename (col1, col2, ...)  
    VALUES (val1, val2, ...),  
           :  
           (val1, val2, val3);
```

- The order of column names must be consistent with the order of values.
- If you are adding a value to every column, you don't have to list them:

```
INSERT INTO tablename VALUES (val1, val2, ...);
```

INSERT: Example

If we try to run the following statements in MySQL, what will the data in the table look like?

```
INSERT INTO `Employee`  
    (`ID`, `Name`, `Salary`)  
VALUES (2, 'Mary', 26000);
```

```
INSERT INTO Employee  
    (Name, ID)  
VALUES ('Mary', 2);
```

```
INSERT INTO Employee  
VALUES (2, 'Mary', 26000),  
       (3, 'Max', 19000);
```

Employee

ID	Name	Salary

INSERT: Example

If we try to run the following statements in MySQL, what will the data in the table look like?

```
INSERT INTO `Employee`  
  (`ID`, `Name`, `Salary`)  
VALUES (2, 'Mary', 26000);
```

```
INSERT INTO Employee  
  (Name, ID)  
VALUES ('Mary', 2);
```

```
INSERT INTO Employee  
VALUES (2, 'Mary', 26000),  
       (3, 'Max', 19000);
```

Employee

ID	Name	Salary
2	Mary	26000
2	Mary	
2	Mary	26000
3	Max	19000

UPDATE

- Changes values in specified rows based on WHERE conditions

```
UPDATE table-name  
    SET col1 = val1 [, col2 = val2...]  
    [WHERE condition]
```

- All rows where the condition is true have the columns set to the given values.
- If no condition is given all rows are changed.
- Values are constants or can be computed from columns.
- Details of the WHERE clause will be taught later.

UPDATE: Example

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Anne	22000

```
UPDATE Employee
SET
    Salary = 15000,
    Name = 'Jane'
WHERE ID = 4;
```

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Jane	15000

```
UPDATE Employee
SET Salary =
    Salary * 1.05;
```

Employee

ID	Name	Salary
1	John	26250
2	Mary	27300
3	Mark	18900
4	Anne	23100

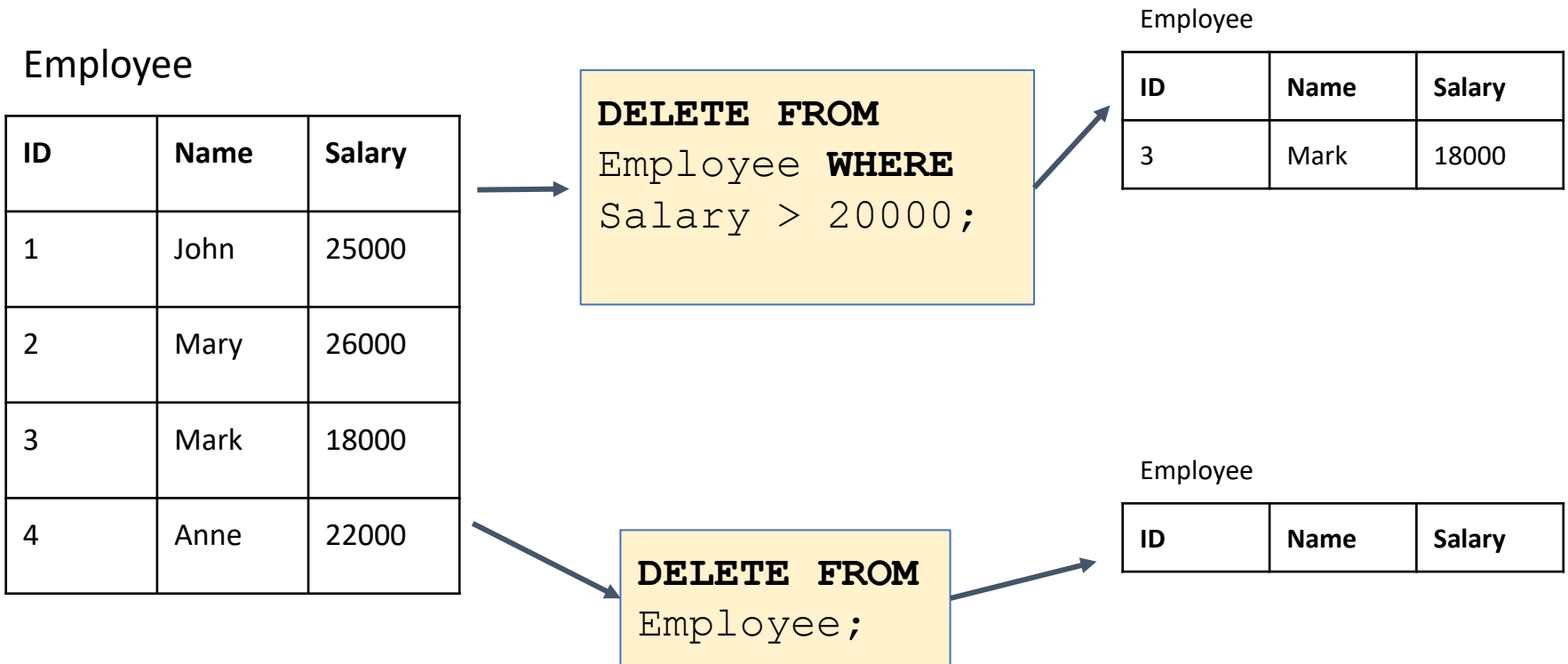
DELETE

- Removes all rows, or those which satisfy a condition

```
DELETE FROM  
    table-name  
    [WHERE condition]
```

- If no condition is given then ALL rows are deleted.

DELETE: Example



Final Example

- Convert the following relation into SQL query.
- Choose appropriate data types for columns.

Branch

branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

- Then update all “B005”s to “B009”
- Delete all tuples that has “B003”

Table Constraints

Domain, Primary key, unique key, foreign key will be covered in the next lecture.

Table Constraints

- Table constraints can be defined when creating tables.
- But you can also add constraints to an existing table.
 - Will be mentioned later.

```
CREATE TABLE name (  
    col-name datatype [col-options],  
    :  
    col-name datatype [col-options],  
    [constraint-1],  
    :  
    [constraint-n]  
);
```

Syntax of Constraints

- General Syntax: **CONSTRAINT** *name* **TYPE** *details*;
- Constraint name is created so that later this constraint can be removed by referring to its name.
 - If you don't provide a name, one will be generated.
- MySQL provides following constraint types
 - **PRIMARY KEY**
 - **UNIQUE**
 - **FOREIGN KEY**
 - **INDEX**
- Most of them can be defined along with a column or separately, see the next few slides.

Domain Constraints

- You can limit the possible values of an attribute by adding a **domain constraint**.
- A domain constraint can be defined along with the column or separately:

```
CREATE TABLE People (  
    id INTEGER PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    sex CHAR NOT NULL CHECK (sex IN ('M', 'F')),  
    CONSTRAINT id_positive CHECK (id > 0)  
);
```

- Supported in MySQL 8.0.16+
- In earlier versions, these constraints will be ignored.

UNIQUE

- Usage:

```
CONSTRAINT name UNIQUE (col1, col2, ...)
```

- A table can have multiple UNIQUE keys defined.

The following unique keys are different:

- One unique key (a, b, c):
 - Tuples (1, 2, 3) and (1, 2, 2) are allowed
- Separate unique keys (a) (b) (c):
 - Tuples (1, 2, 3) and (1, 2, 2) are NOT allowed

Primary Key

- Usage:

```
CONSTRAINT name PRIMARY KEY (col1, col2 ...)
```

- Constraint name for a primary key is actually ignored by MySQL, but works in other databases.
- **PRIMARY KEY** also automatically adds **UNIQUE** and **NOT NULL** to the relevant column definition
- Extended reading:
 - [Difference](#) between Primary Key and Unique.
 - More [underlying mechanism](#).







Example

```
CREATE TABLE Branch (  
    branchNo CHAR(4) ,  
    street VARCHAR(100) ,  
    city VARCHAR(25) ,  
    postcode VARCHAR(7) ,  
    CONSTRAINT branchUnique UNIQUE  
        (postcode) ,  
    CONSTRAINT branchPK  
        PRIMARY KEY (branchNo)  
);
```

Example (Alternative Way)

```
CREATE TABLE Branch2 (  
    branchNo CHAR(4) PRIMARY KEY,  
    street VARCHAR(100),  
    city VARCHAR(25),  
    postcode VARCHAR(7) UNIQUE  
);
```

- The primary key and unique key will automatically be assigned with constraint names.

	Name	Owner	Type
 Columns			
 Constraints	 PRIMARY	Branch2	PRIMARY KEY
	 postcode	Branch2	UNIQUE KEY
 Foreign Keys			
 References			

Question

- Will the following piece of code work?
- Assume that columns (booking_time, guest_id) should be set as the primary key.

```
CREATE TABLE room_booking (  
    booking_time DATETIME primary key,  
    room_number INT,  
    guest_id VARCHAR(100) primary key,  
    comments TEXT  
);
```