MySQL 数据库

目录：

1. 数据库介绍

数据库，就是能够存储和管理“大量数据”的一种软件系统的统称。

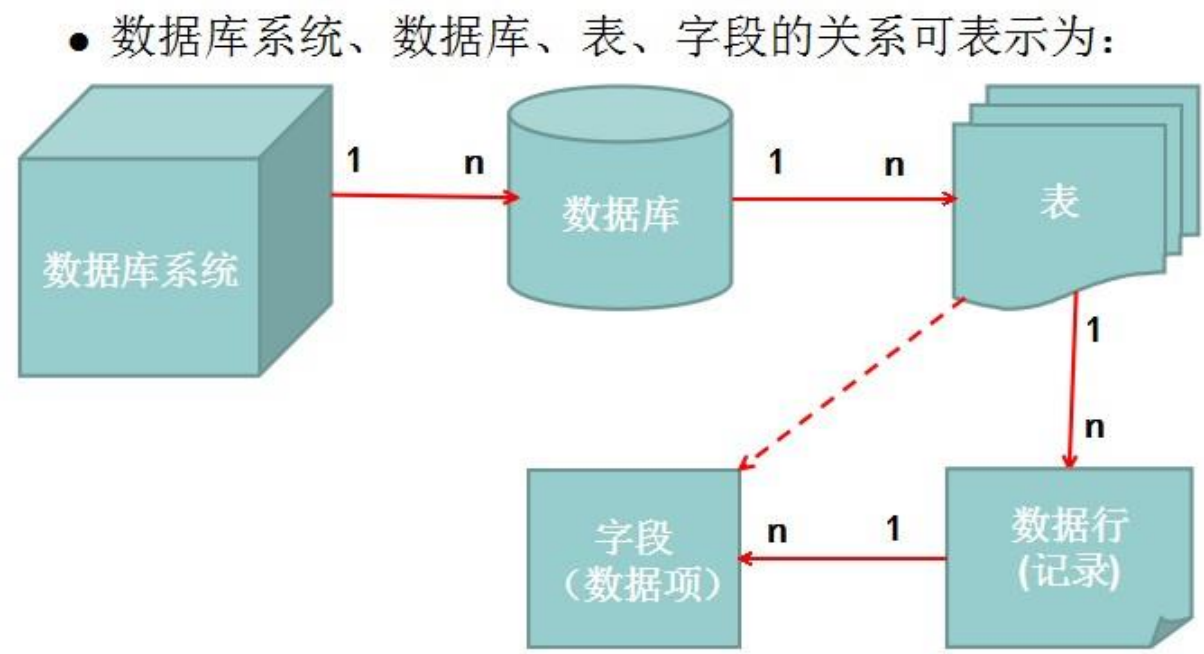
1.1. 主流数据库

主流数据库包括：MS SQL Server， Oracle，DB2，Informix， Sybase 等。

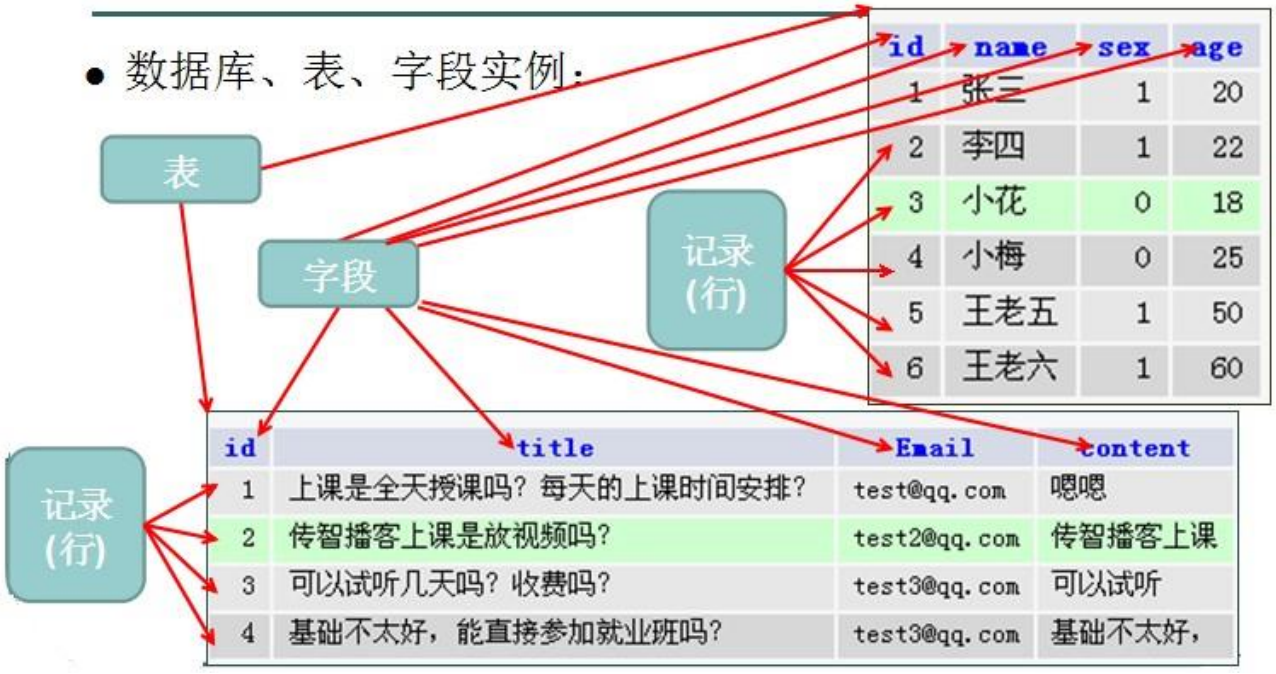
他们都是被称为“关系数据库”的一种遵循 sql 标准的软件产品。

1.2. MySQL 数据库概览

MySQL 数据库的基本结构如下所示：



其中，实际表的数据和结构如下所示：



对应的几个名词（单词）为：

DBMS： 数据库（管理）系统，是我们“安装”而得到的。

DB，DataBase： 数据库，一个数据库系统中可以存放多个数据库。
通常一个项目（网站）使用一个数据库来存储其中的数据。

table：表，一个数据库中可以存放多个表。

row：行，指一行数据，一个表中可以有很多行。

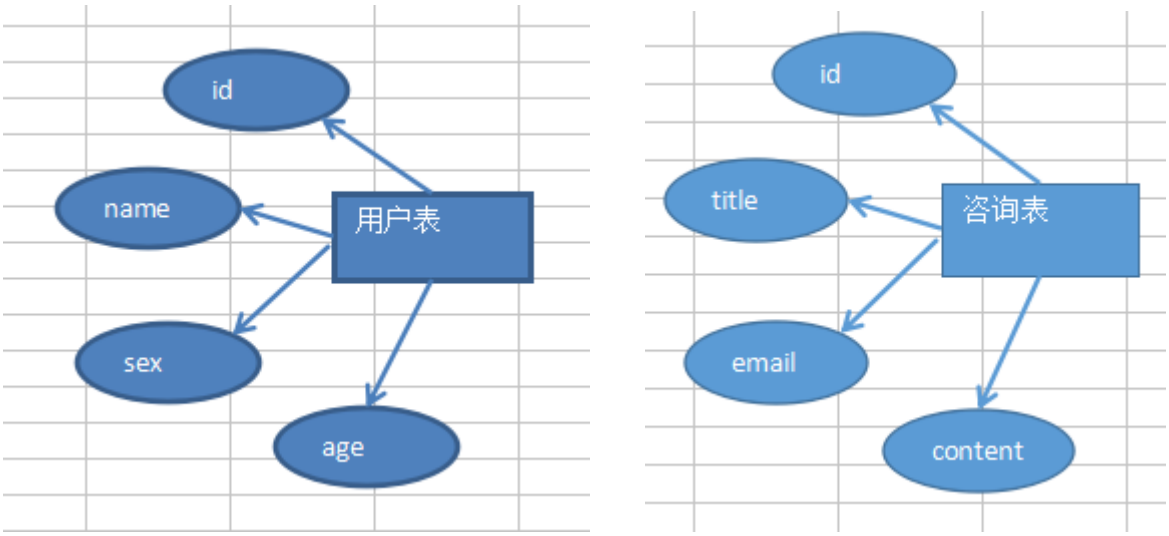
record：记录，也是指一行数据。

column：列，指一列数据，一个表可以有若干列。

field：字段（列名），指数据表中的一列的名称（类似表头），一个表可以有若干字段。

1.3. 关系数据库

关系数据库是指基于关系模型而设计的数据库系统。
所谓关系，其实就是指一个二维表（table）（有行有列）。
一行有多个数据，就表示这多个数据是具有内在关系的（同属一个“实体”）。
比如，上述两个表，可以用“E-R”图（实体-关系图）表示如下：

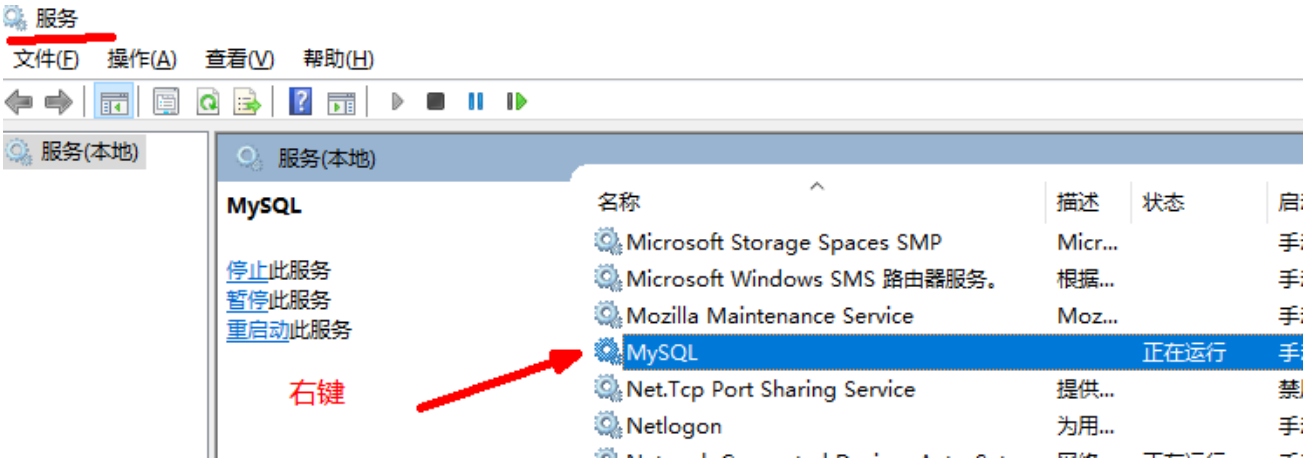


2. 访问 mysql 数据库服务器

通常，我们要把数据库理解为“你用，还是不用，它就在那里！”

2.1. 开启/关闭数据库服务

2.1.1. 通过系统服务

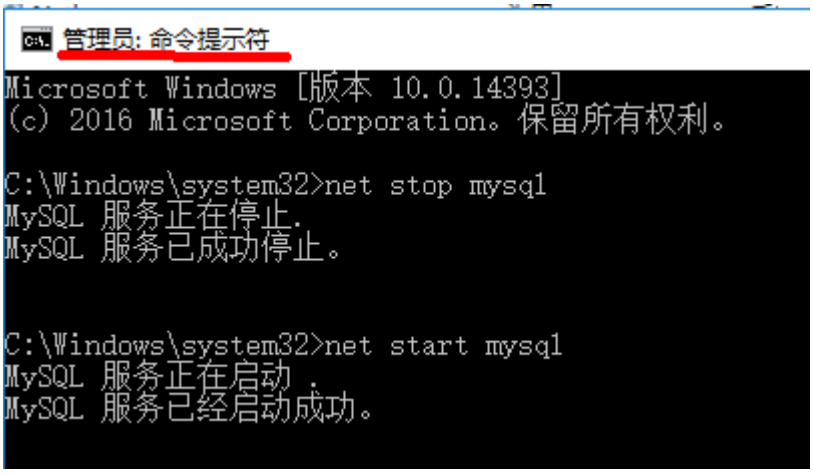


2.1.2. 通过命令行方式

在管理员模式下运行 cmd，执行如下命令：

net start mysql

net stop mysql



2.2. 客户端连接数据库服务器

数据库就在那里！连，还是不连，就看你了！

任何连接或访问数据库的“软件/工具”，都可以称为“客户端”。

2.2.1. 命令行连接数据库

连接（进入）数据库命令：

mysql -h 主机地址 -u 用户名 -p

特别注意：cmd 中登录后，请立即使用“set names gbk;” 语句来设定连接编码。

表示当前连接到数据库的“客户端”的字符编码是 gbk（固定的，不可更改）。

退出数据库命令：

quit;

或

exit;

或

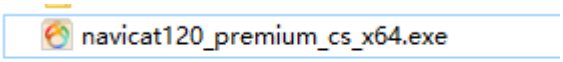
\q;

```
C:\Users\ldh>mysql -hlocalhost  -uroot  -p
Enter password: ***
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.5.59 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> set names gbk;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

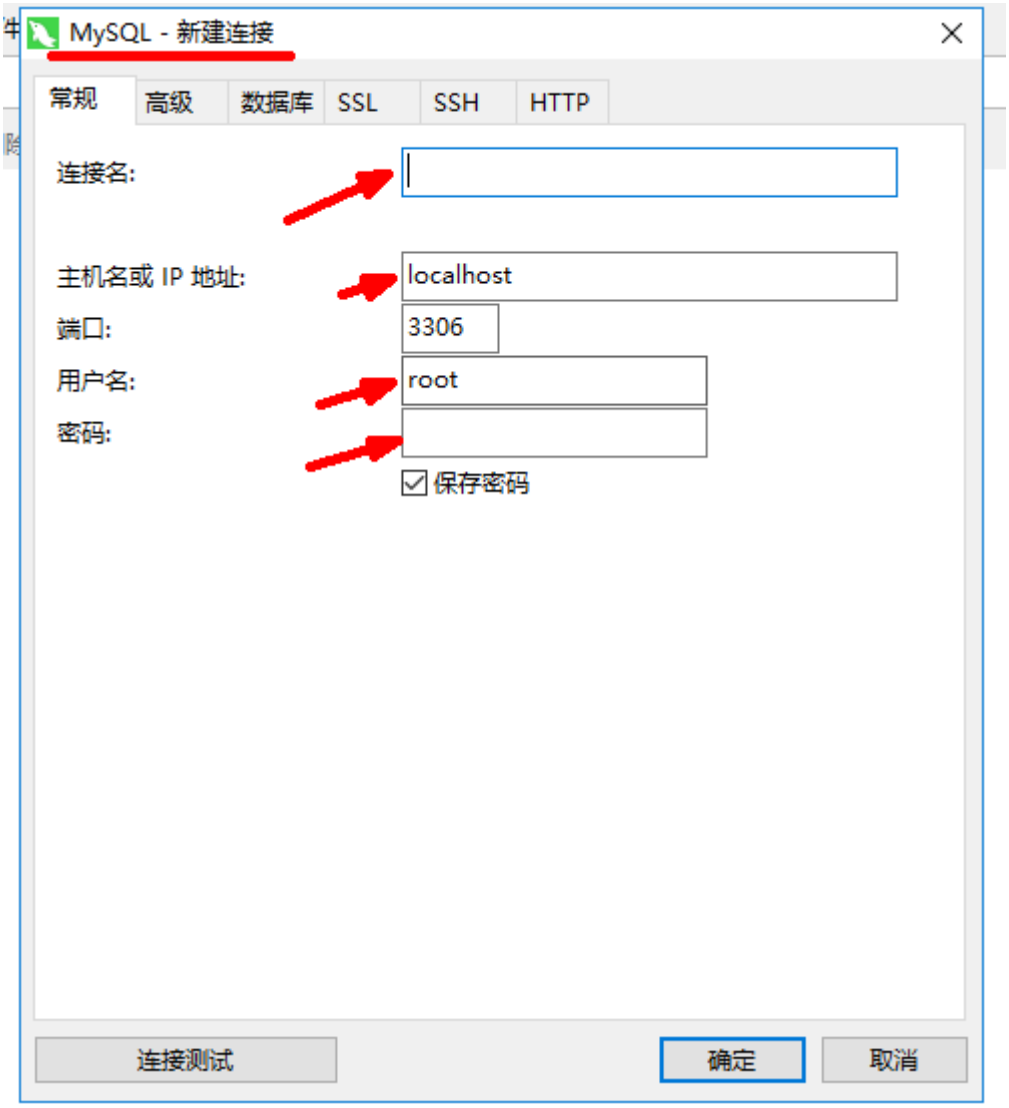
cmd中登录后，请立即先执行这一条语句

登录成功后的命令行前缀符号

2.2.2. navicat 软件连接数据库



安装它，然后打开软件：



2.2.3. phpmyadmin“网站”连接数据库

安装（配置）该站点：

1， hosts 文件中设定域名解析： www.myadmin69.com

2， 拷贝网站文件到指定目录： bj-php-69/myadmin/

3， httpd-vhost.conf 文件中设定站点：

```
<VirtualHost  *:80>
      .....
</VirtualHost>
```

```
httpd-vhosts.conf
94
95
96  #站点3：（第一个站点，被称为默认站点）
97  <VirtualHost *:80>
98      ServerName  www.myadmin69.com
99      DocumentRoot  "H:\itcast\class\bj-php-69\MyAdmin69"
100     <Directory  "H:\itcast\class\bj-php-69\MyAdmin69" >
101         #允许列出目录
102         Options Indexes
103         #允许权限覆盖
104         AllowOverride  All
105         #允许所有访问
106         Require  all  granted
107     </Directory>
108     DirectoryIndex  index.php  index.html
109 </VirtualHost>
```

3. 数据库操作

3.1. 查看所有数据库

语句形式：

show databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.00 sec)
```

一个错误提示：

```
mysql> show datebases;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server
 for the right syntax to use near 'datebases' at line 1
mysql>
```

3.2. 创建新数据库

语句形式：

create database 数据库名 [charset 字符集名称] [collate 校对规则名];

字符集名类似这些： utf8, gbk, gb2312, big5, ascii 等。推荐用 utf8。

校对规则名：通常都不用写，而是使用所设定字符集的默认校对规则。

校对规则的含义：

就是一个字符集中的每个字符的“排序规则”。

对于英文，很简单，就是按英文单词的字母顺序。

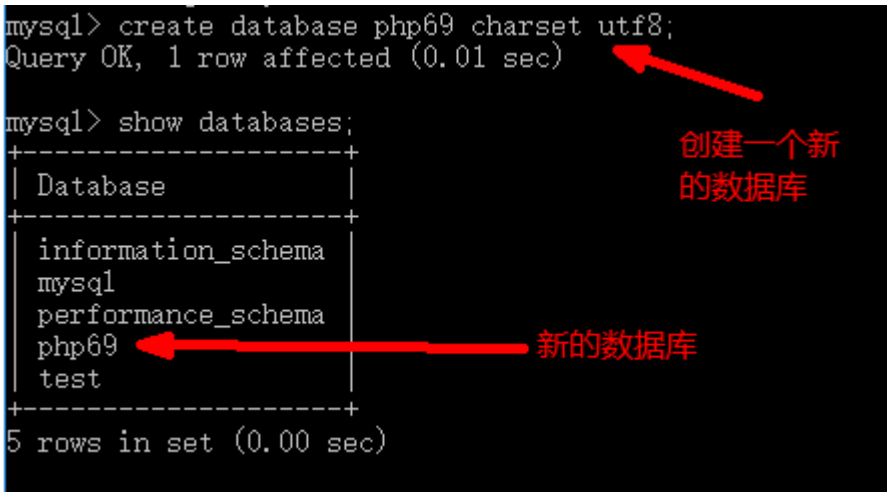
对于中文，或其他一些亚洲语言，就会面临问题：两个字的顺序，到底谁先谁后（谁大谁小）呢？

比如：“传”和“智”，有这样的可能排序方式：

按拼音：“传”在前（更小），“智”在后（更大）；

按笔顺（横竖撇捺折）：“智”在前（更小），“传”在后（更大）；

按编码：肯定一个大一个小（具体未知）；



查看可用的字符集：

show charset;

查看可用的校对规则：

show collation;

3.3. 查看数据库创建信息

语句形式：

show create database 数据库名;

结果其实就是能看到指定数据的完整创建语句（含默认值的选项，比如 charset， collate）。

3.4. 删除现有数据库

语句形式：

drop database 数据库名;

3.5. 修改现有数据库

修改数据库，其实只是修改数据库的字符编码或校对规则。
其实一般都不需要修改。

语句形式：

alter database 数据库名 charset 新的字符集名称 collate 新的校对规则名 ；

```
mysql> alter database php69 charset gbk;
Query OK, 1 row affected (0.00 sec)

mysql> alter database php69 charset utf8;
Query OK, 1 row affected (0.00 sec)
```

3.6. 选择（使用）某个数据库

一个项目中，具体进行有关数据操作（增删改查）之前，都需要先“选择/进入”该数据库。

语句形式：

use 数据库名;

```
mysql> use php69;
Database changed
mysql>
```

总结有关数据库的常规操作：

创建数据库：

create database 数据库名 charset 编码名（推荐 utf8）；

显示所有数据库：

show databases ；

显示某个数据库的创建语句：

show create database 数据库名；

删除数据库：

drop database 数据库名；

修改某个数据库（的字符集和排序规则）

alter database 数据库名 charset 新字符集名称 collate 新校对规则名；

使用（进入/选择）数据库：

use 数据库名；

4. 数据表操作

“数据库”只是一个外壳，除了有个数据库名称和字符集设定，基本就没有别的信息了。
数据表才是存储（装载）数据的具体“容器”。
我们需要创建不同的表来存储不同的数据。

4.1. 创建数据表初步

语句形式：

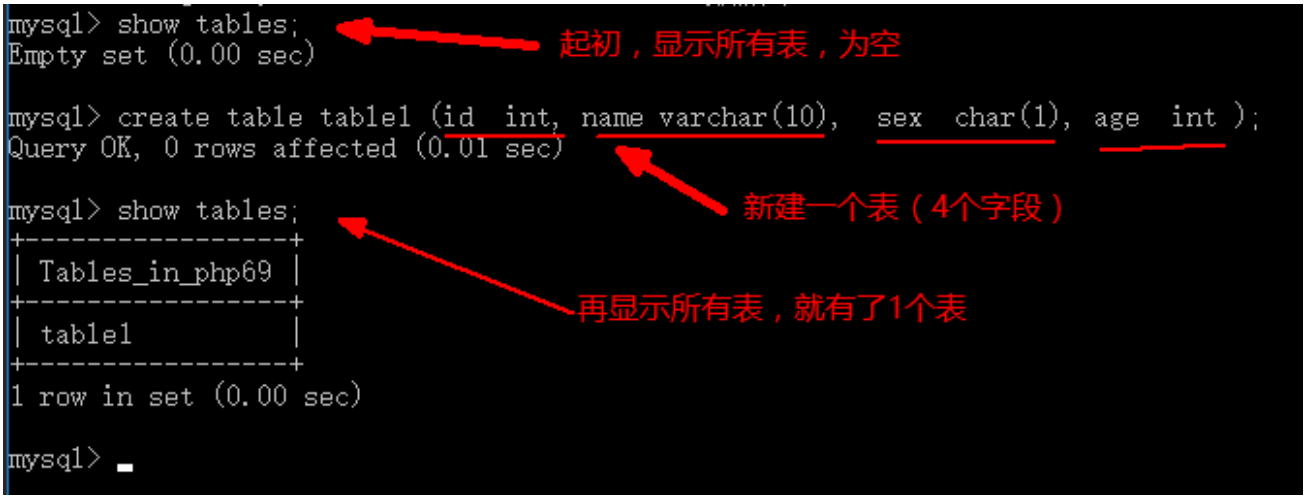
create table 数据表名 （字段 1， 字段 2， ... ）[charset=字符集] [engine=表类型];

其中：

字段的形式为： 字段名 字段类型 [字段属性...]

字符集包括：utf，gbk，gb2312，big5 等等，默认是数据库的字符集，可以不写。

表类型包括：InnoDB， MyIsam，BDB，等，默认是 InnoDB，可以不写。



4.2. 查看所有数据表

语句形式：

show tables；

```
mysql> create  table  info (
    -> id  int,
    -> title  varchar(50),
    -> email varchar(20),
    -> content  text
    -> )
    -> charset utf8
    -> engine MyIsam ;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+----------------+
| Tables_in_php69 |
+----------------+
| info           |
| table1         |
+----------------+
2 rows in set (0.00 sec)
```

再建立一个表

可以看到又多了一个表

4.3. 查看数据表结构

语句形式：

desc 表名;

所谓数据表的结构，其实就是一个表的每个字段的具体信息。

```
mysql> desc  table1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| name  | varchar(10) | YES  |     | NULL    |       |
| sex   | char(1)     | YES  |     | NULL    |       |
| age   | int(11)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

是否可为空　默认值　类型　这里表名当前这个表table1有4个字段

在来一个：

```
mysql> desc info;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int(11)     | YES  |     | NULL    |       |
| title   | varchar(50) | YES  |     | NULL    |       |
| email   | varchar(20) | YES  |     | NULL    |       |
| content | text        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

4.4. 查看数据表的创建语句

语句形式：

show create table 表名；

SQL 查询： show create table info;
记录： 1

Table	Create Table
info	CREATE TABLE `info` (`id` int(11) DEFAULT NULL, `title` varchar(50) DEFAULT NULL, `email` varchar(20) DEFAULT NULL, `content` text) ENGINE=MyISAM DEFAULT CHARSET=utf8

（在 phpmyadmin 的界面执行的结果）

4.5. 删除数据表

语句形式：

drop table 表名；

```
mysql> drop table tab1;
Query OK, 0 rows affected (0.01 sec)
```

有关表的基本操作的总结：

建表：

create table 表名 (
字段名 字段类型 [字段属性],
字段名 字段类型 [字段属性],
.....
) [charset=编码名称] [engine=表类型名称]；
表类型名称可用的也就几个，比如： InnoDB（默认的）, MyIsam, BDB, memory,

显示所有表：

show tables;

显示某个表的创建语句：

show create table 表名；

显示某个表的结构：

desc 表名；

删除表：

drop table 表名；

4.6. 修改数据表

修改数据表主要是修改表名，添加字段，修改字段，删除字段，修改表的字符集；

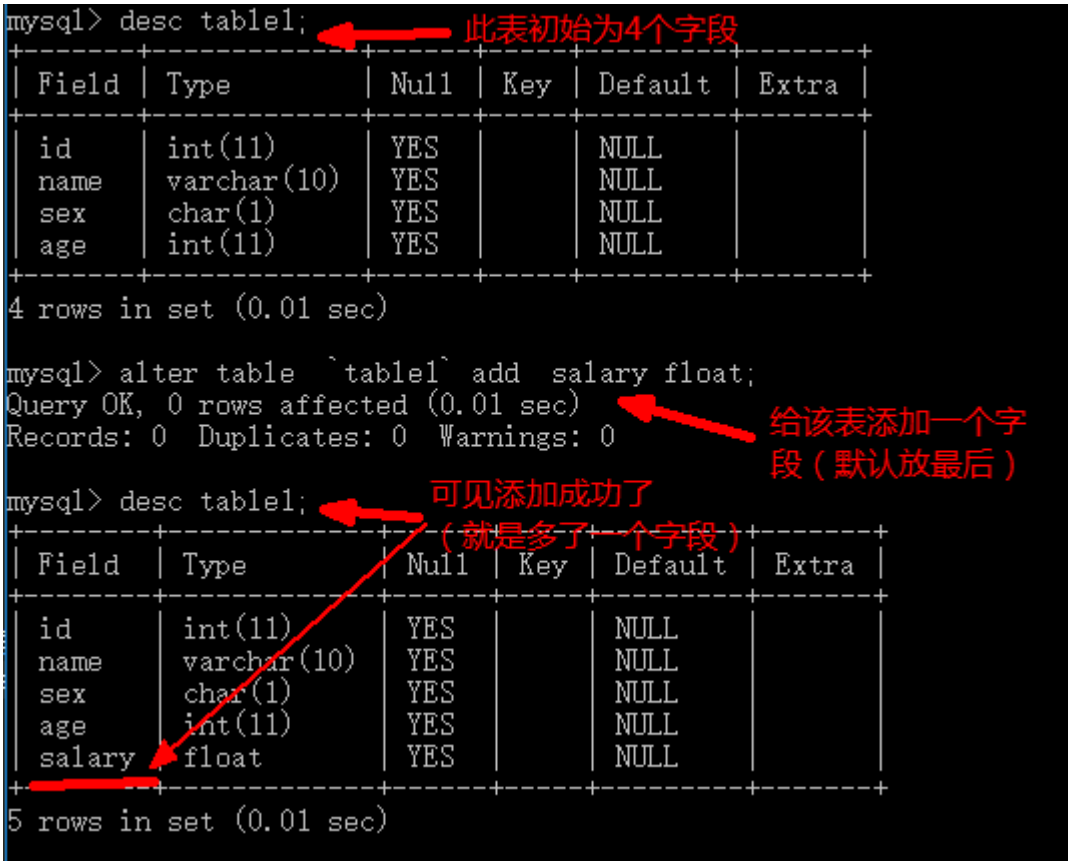
4.6.1.1. 添加字段：

语句形式：

alter table 表名 add 字段名 字段类型 [字段属性...] [after 某字段名 或 first]；

after 某字段名：意思是，新加的字段，放在该现有字段的后面；

first：表示新加的字段放在第一位（最前面）




```
mysql> desc table1;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| id    | int(11)    | YES  |     | NULL    |       |
| name  | varchar(10)| YES  |     | NULL    |       |
| sex   | char(1)    | YES  |     | NULL    |       |
| age   | int(11)    | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql> alter table `table1` add salary float;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc table1;
+--------+------------+------+-----+---------+-------+
| Field  | Type       | Null | Key | Default | Extra |
+--------+------------+------+-----+---------+-------+
| id     | int(11)    | YES  |     | NULL    |       |
| name   | varchar(10)| YES  |     | NULL    |       |
| sex    | char(1)    | YES  |     | NULL    |       |
| age    | int(11)    | YES  |     | NULL    |       |
| salary | float      | YES  |     | NULL    |       |
+--------+------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

```
mysql> alter table  table1 add edu varchar(5) after age;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc table1;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | YES  |     | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| sex    | char(1)     | YES  |     | NULL    |       |
| age    | int(11)     | YES  |     | NULL    |       |
| edu    | varchar(5)  | YES  |     | NULL    |       |
| salary | float       | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

4.6.1.2. 修改字段：

语句形式：

alter table 表名 change 旧字段名 新字段名 字段类型 [字段属性...];

如果不修改字段名，而只修改字段的其他信息，则可以使用：

alter table 表名 modify 要修改的字段名 字段类型 [字段属性...];

```
mysql> alter  table table1  change salary gongzi int default 0;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc table1;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | YES  |     | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| sex    | char(1)     | YES  |     | NULL    |       |
| age    | int(11)     | YES  |     | NULL    |       |
| edu    | varchar(5)  | YES  |     | NULL    |       |
| gongzi | int(11)     | YES  |     | 0       |       |
+--------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

```
mysql> alter  table  table1  modify edu varchar(10) after gongzi;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc table1;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | YES  |     | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| sex    | char(1)     | YES  |     | NULL    |       |
| age    | int(11)     | YES  |     | NULL    |       |
| gongzi | int(11)     | YES  |     | 0       |       |
| edu    | varchar(10) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

4.6.1.3. 删除字段：

语句形式：

alter table 表名 drop 要删除的字段名；

4.6.1.4. 修改表名：

语句形式：

alter table 表名 rename 新的表名；

```
修改表名：
alter table  table1  rename user;
```

4.6.1.5. 修改字符集：

语句形式：

alter table 表名 charset=新的字符集；

修改表语句的总结：

添加一个字段：

alter table 表名 add 一个字段的信息；

一个字段的信息的意思是：字段名 字段类型 [字段属性]

改掉一个字段：

alter table 表名 change 旧的字段名 新的字段信息；

修改一个字段：

alter table 表名 modify 要修改的字段名 新的类型 [新的属性]

删除字段：

alter table 表名 drop 要删除的字段名；

修改表名：

alter table 表名 rename 新的表名；

修改字符集：

alter table 表名 charset=新的字符集；

5. 数据操作初步

数据都是存储在数据表中。
数据的基本操作有 4 种：增（插入 insert），删（删除 delete），改（修改 update），查（查询 select）。
即所谓的 CRUD 操作： Create（创建）， Retrieve（获取）， Update（更新）， Delete（删除）。

5.1. 插入数据

语句形式：

insert into 表名 (字段名 1， 字段名 2， ...) values (数据 1， 数据 2， ... ）；

说明：

1，字段名和数据是“一一对应”的，包括：数量一致，顺序一致，类型匹配。

2，对于要写入的数据，字符串和时间日期类型，要用单引号引起来。

3，可以省略“字段列表”部分，此时就需要给出跟字段数量一样多的数据，类似这样：

① insert into 表名 values (数据 1， 数据 2， ... ）；

```
mysql> insert into info (id, title, email, content)
    -> values(1, '今天我们可以不上自习吗？', 'tianxiang@qq.com', '不可以！');
Query OK, 1 row affected (0.00 sec)

mysql> insert into info (id, title, email, content)
    -> values(1, '明天我们要上 自习吗？', '1234567@qq.com', '必须！');
Query OK, 1 row affected (0.00 sec)

mysql> select * from  info;
+------+-------------------------+-----------------+----------+
| id   | title                   | email           | content  |
+------+-------------------------+-----------------+----------+
|    1 | 今天我们可以不上自习吗？ | tianxiang@qq.com | 不可以！ |
|    1 | 明天我们要上 自习吗？   | 1234567@qq.com   | 必须！   |
+------+-------------------------+-----------------+----------+
2 rows in set (0.00 sec)
```

```
mysql> insert into info (id, title, email)
    -> values(3, '标题3 自习吗？', '33333@qq.com');
Query OK, 1 row affected (0.00 sec)

mysql> select * from info;
+------+-------------------------+-----------------+----------+
| id   | title                   | email           | content  |
+------+-------------------------+-----------------+----------+
|    1 | 今天我们可以不上自习吗？                    | tianxiang@qq.com | 不可以!        |
|    1 | 明天我们要上 自习吗？                  | 1234567@qq.com   | 必须!        |
|    3 | 标题3 自习吗？                  | 33333@qq.com     | NULL     |
+------+-------------------------+-----------------+----------+
3 rows in set (0.00 sec)

mysql> insert into info
    -> values(4, '标题4 自习吗？', '4444@qq.com', '回复内容4');
Query OK, 1 row affected (0.00 sec)

mysql> select * from info;
+------+-------------------------+-----------------+-----------+
| id   | title                   | email           | content   |
+------+-------------------------+-----------------+-----------+
|    1 | 今天我们可以不上自习吗？                    | tianxiang@qq.com | 不可以!         |
|    1 | 明天我们要上 自习吗？                  | 1234567@qq.com   | 必须!         |
|    3 | 标题3 自习吗？                  | 33333@qq.com     | NULL      |
|    4 | 标题4 自习吗？                  | 4444@qq.com      | 回复内容4        |
+------+-------------------------+-----------------+-----------+
4 rows in set (0.00 sec)
```

5.2. 查询数据

语句形式：

select 字段名 1， 字段名 2， ... from 表名 [where 条件]；

说明：

1，select 后的字段列表用于设定要从表中取出的哪些字段的值。

2，select 后可以只使用一个“*”号表示取出该表中所有字段的值。

3，where 条件表示取出的数据所应满足的条件，比如：

① where id < 10 表示取出那些字段 id 的值小于 10 的行。

② where age > 60 表示取出那些字段 age 的值大于 60 的行

4，where 条件可以不写，就取出所有行的数据。

```
mysql> select pro_id, pro_name  from product where pro_id<10 and price > 5000;
+--------+-----------------------------------+
| pro_id | pro_name                          |
+--------+-----------------------------------+
|      4 | 联想（Lenovo）14.0英寸笔记本电脑                |
|      5 | 索尼（SONY）13.3英寸触控超极本                 |
+--------+-----------------------------------+
2 rows in set (0.00 sec)
```

5.3. 删除数据

语句形式：

delete from 表名 [where 条件]；

说明：

1， 删除数据指的是删除表的某些行，比如原来有 10 行，可以将其中的 3 行删除，则剩下 7 行。

2， where 条件表示删除数据所应满足的条件，含义跟 select 中的一样。

3， where 条件可以不写，如果不写，则会删除所有数据——通常都不会这么用。

```
mysql> delete from user_info;
Query OK, 6 rows affected (0.00 sec)

mysql> delete from order_goods where id>6 and order_id = 4;
Query OK, 1 row affected (0.00 sec)
```

5.4. 修改数据

语句形式：

update 表名 set 字段名 1 = 新值 1， 字段名 2=新值 2， ... [where 条件]；

说明：

1， 修改数据指的是修改表的某些行的某些字段。

2， where 条件表示修改数据所应满足的条件，含义跟 select 中的一样。

3， where 条件可以不写，如果不写，则会修改所有数据——通常都不会这么用。

```
mysql> select * from info;
+------+-------------------------+-----------------+-----------+
| id   | title                   | email           | content   |
+------+-------------------------+-----------------+-----------+
|    1 | 今天我们可以不上自习吗？ | tianxiang@qq.com | 不可以!   |
|    1 | 明天我们要上 自习吗？    | 1234567@qq.com   | 必须!     |
|    3 | 标题3 自习吗？           | 33333@qq.com     | NULL      |
|    4 | 标题4 自习吗？           | 4444@qq.com      | 回复内容4 |
+------+-------------------------+-----------------+-----------+
4 rows in set (0.00 sec)

mysql> update info set  content = '新的修改的内容' where id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from info;
+------+-------------------------+-----------------+----------------+
| id   | title                   | email           | content        |
+------+-------------------------+-----------------+----------------+
|    1 | 今天我们可以不上自习吗？ | tianxiang@qq.com | 不可以!        |
|    1 | 明天我们要上 自习吗？    | 1234567@qq.com   | 必须!          |
|    3 | 标题3 自习吗？           | 33333@qq.com     | 新的修改的内容 |
|    4 | 标题4 自习吗？           | 4444@qq.com      | 回复内容4      |
+------+-------------------------+-----------------+----------------+
4 rows in set (0.00 sec)
```

```
mysql> select * from info;
+------+--------------------------+-----------------+----------------+
| id   | title                    | email           | content        |
+------+--------------------------+-----------------+----------------+
|    1 | 今天我们可以不上自习吗？                   | tianxiang@qq.com | 不可以!
|    1 | 明天我们要上 自习吗？                  | 1234567@qq.com   | 必须!
|    3 | 标题3 自习吗？                 | 33333@qq.com     | 新的修改的内容
|    4 | 标题4 自习吗？                 | 4444@qq.com      | 回复内容4
+------+--------------------------+-----------------+----------------+
4 rows in set (0.00 sec)

mysql> update info set title='新的标题',  content = '新的内容' where id > 2;
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from info;
+------+--------------------------+-----------------+----------+
| id   | title                    | email           | content  |
+------+--------------------------+-----------------+----------+
|    1 | 今天我们可以不上自习吗？                   | tianxiang@qq.com | 不可以!
|    1 | 明天我们要上 自习吗？                  | 1234567@qq.com   | 必须!
|    3 | 新的标题                   | 33333@qq.com     | 新的内容
|    4 | 新的标题                   | 4444@qq.com      | 新的内容
+------+--------------------------+-----------------+----------+
4 rows in set (0.00 sec)
```

总结：

系统级操作：

服务器的启停

mysql 服务，通过系统服务来操作，或:

net start mysql

net stop mysql

登录系统：

mysql -hlocalhost -uroot -p

退出：

quit

exit

客户端：

cmd 方式：

phpmyadmin 方式：

navicat 方式：

库操作：

建库：

create database 数据库名 charset utf8 [collate 校对规则];

show charset; //显示所有可用的编码（字符集）

show collation; //显示所有可用的校对规则（排序规则）

删除数据库：

drop database 数据库名;

show databases;

show create database 数据库名；
use 数据库名；

表操作：

建表：

create table 表名 (
字段名 1 类型 [属性],
字段名 2 类型 [属性],
.....
)
charset = 编码 engine =表类型名称
表类型有：InnoDB, MyIsam, BDB, Memory
表类型也叫做“存储引擎”

特点	Myisam	InnoDB	BDB	Memory	Archive
批量插入的速度	高	低	高	高	非常高
事务安全		支持	支持		
全文索引	支持	5.6版本支持			
锁机制	表锁	行锁	页锁	表锁	行锁
存储限制	没有	64TB	没有	有	没有
B树索引	支持	支持	支持	支持	
哈希索引		支持		支持	
集群索引		支持			
数据缓存		支持		支持	
索引缓存	支持	支持		支持	
数据可压缩	支持				支持
空间使用	低	高	低	N/A	非常低
内存使用	低	高	低	中等	低
支持外键		支持			

desc 表名；
show create table 表名；
show tables;
drop table 表名；
alter table 修改表有如下可修改项：
alter table 表名 add 新的字段；
alter table 表名 change 改掉字段；
alter table 表名 modify 修改字段；
alter table 表名 rename 新的表名；
alter table 表名 charset = 新的字符集名；

数据操作：

增：

insert into (字段列表) values (值列表);

删：

delete from 表名 where 条件；

改：

update 表名 set 字段名 = 新的值，..... where 条件

查：

select 字段列表 from 表名 where 条件。

```
介绍建表语句中的2个特别属性：
create table user2 (
        id  int auto_increment  primary key,
        name varchar(10),
        sex  char(1),
        age  int ) charset utf8;
-- auto_increment用于整数类型，能让该字段按顺序“自动增长”
-- primary key用于表示（限定）该字段的值不能重复，并且作为“主要关键字”

insert into user2  values( null, '张三', '男', 18 )
insert into user2  values( null, '李四', '女', 20 )

insert into user2 (name, sex, age) values( '李5', '男', 21 )
insert into user2 (id,name, sex, age) values( 1, '李6', '男', 21 );
```

```
mysql> create table user2 (
    -> id  int auto_increment  primary key,
    -> name varchar(10),
    -> sex  char(1),
    -> age  int ) charset utf8;
Query OK, 0 rows affected (0.01 sec)

mysql> desc user2;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | int(11)    | NO   | PRI | NULL    | auto_increment |
| name  | varchar(10)| YES  |     | NULL    |                |
| sex   | char(1)    | YES  |     | NULL    |                |
| age   | int(11)    | YES  |     | NULL    |                |
+-------+------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

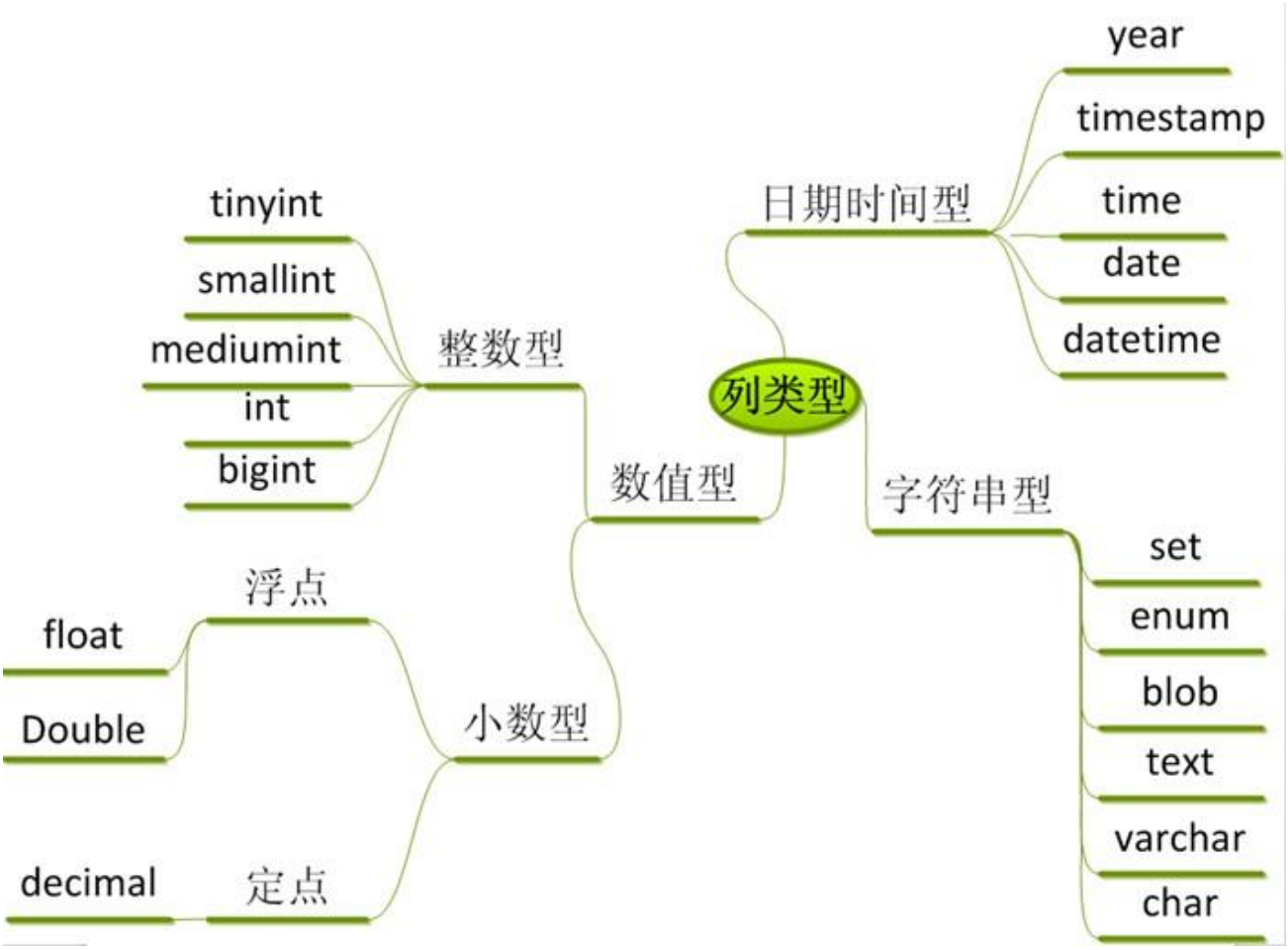
```
mysql> insert into user2  values( null, '张三', '男', 18 );
Query OK, 1 row affected (0.00 sec)

mysql> select * from user2;
+----+------+------+------+
| id | name | sex  | age  |
+----+------+------+------+
|  1 | 张三     | 男     |   18 |
+----+------+------+------+
1 row in set (0.00 sec)

mysql> insert into user2  values( null, '李四', '女', 20 );
Query OK, 1 row affected (0.01 sec)

mysql> select * from user2;
+----+------+------+------+
| id | name | sex  | age  |
+----+------+------+------+
|  1 | 张三     | 男     |   18 |
|  2 | 李四     | 女     |   20 |
+----+------+------+------+
2 rows in set (0.00 sec)

mysql> insert into user2 (name, sex, age) values( '李5', '男', 21 );
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into user2 (name, sex, age) values( '李5', '男', 21 );
Query OK, 1 row affected (0.00 sec)

mysql> select * from user2;
+----+------+------+------+
| id | name | sex  | age  |
+----+------+------+------+
|  1 | 张三     | 男     |   18 |
|  2 | 李四     | 女     |   20 |
|  3 | 李5      | 男     |   21 |
+----+------+------+------+
3 rows in set (0.00 sec)

mysql> insert into user2 (id,name, sex, age) values( 1, '李6', '男', 21 );
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql>
```

》》》day2

6. MySQL 数据类型

6.1. 数据类型（列类型）总览



可见，mysql 中的数据类型，总体分 3 大类：

数字型：

时间型：

字符型：

其中，在 sql 语句中，数字型数据不需要单引号引起来，而时间型和字符型数据需要用单引号引起来。

6.2. 整型

整型数据类型包括：

tinyint	：	微整型
smallint	：	小整型
mediumint	：	中整型
int	：	整型
bigint	：	大整型

这些不同大小范围的整型信息如下表所示：

类型	所占空间（字节）	带符号		无符号	
		最小值	最大值	最小值	最大值
tinyint	1	-128	127	0	255
smallint	2	-32768	32767	0	65535
mediumint	3	-8388608	8388607	0	16777215
int	4	-2147483648	2147483647	0	4294967295
bigint	8	-9223372036854775808	9223372036854775807	0	18446744073709551615

默认整数类型是带符号的，即可以有正负值，比如：

```
create table zhengxing1(num1 int, num2 tinyint);
```

此时，num1 和 num2 中都可以存储负数（但都不能超出范围）

不带符号的整数类型设置形式如下：

```
create table zhengxing2(num1 int unsigned, num2 tinyint unsigned);
```

6.3. 小数型

小数类型分为浮点小数和定点小数。

6.3.1. 浮点小数

浮点小数是“不精确的小数”，包括 float 和 double。

float：

占用 4 字节存储空间，可称为“单精度浮点数”，约 7 位有效数字。

double：

占用 8 字节存储空间，可称为“双精度浮点数”，约 17 位有效数字。

6.3.2. 定点小数

定点小数是“精确的小数”——它通过内部技巧，突破了“有些小数无法用二进制精确表示”的局限。

其设定方式通常是这样的： decimal(M, D);

其中 M 表示该小数的总的有效位数（最大 65），D 表示该小数的小数点后的位数。

演示：

定义三个字段分别为 float、double 和 decimal 类型，并都插入数字“123456789.123456789123456789”，显示结果。

```
mysql> create table xiaoshu1(x1 float, x2 double, x3 decimal(23, 14) );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into xiaoshu1(x1, x2, x3)values(123456789.123456789123456789, 123456789.123456789123456789,123456789.12345
6789123456789);
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> select * from xiaoshu1;
+-----------+-------------------+-------------------------+
| x1        | x2                | x3                      |
+-----------+-------------------+-------------------------+
| 123457000 | 123456789.12345679 | 123456789.12345678912346 |
+-----------+-------------------+-------------------------+
1 row in set (0.00 sec)
```

这里是17位有效数字

而它（decimal）的有效位数，完全是按我们设定的位置来定。

这里只有6位有效数字

数据类型选择示例：

要设置一个字段为“年龄”：应该用tinyint unsigned

要设置一个人的“工资”：double unsigned可以，decimal unsigned

要存成全国人口的编号：int unsigned

6.4. 日期时间型

日期时间类型包括如下几种：

date 类型：

表示日期，格式类似这样：'0000-00-00'

time 类型：

表示时间，格式类似这样：'00:00:00'

datetime 类型：

表示日期时间，格式类似这样：'0000-00-00 00:00:00'

timestamp 类型：

表示“时间戳”，其实就是一个整数数字，该数字是从是“时间起点”到现在为止的“秒数”。

“时间起点”是：1970-1-1 0:0:0

timestamp 类型的字段，无需插入数据，而是会自动取得当前的日期时间（表示当前时刻）。

而且，此类型字段会在数据被更新时，也同样自动取得当前的日期时间（表示修改的时刻）。

特别总结：它在 insert 或 update 某行数据的时候，能够自动获得当前时间。

year 类型：

表示年份，格式为：'0000'

注意：

时间类型的字面值，通常使用单引号引起来

示例：

创建一个表，设定 5 个字段分别为上述类型，并插入相应的数据值后查看结果。

```
mysql> create table shijian1(
    -> t_time time,
    -> t_date date,
    -> t_datetime datetime,
    -> t_timestamp timestamp,
    -> t_year year
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into shijian1(t_time, t_date, t_datetime, t_year)values
    -> ('10:57:50','2018-7-29', '2008-8-8 20:0:0', '2008');
Query OK, 1 row affected (0.00 sec)

mysql> select * from shijian1;
+----------+------------+---------------------+---------------------+--------+
| t_time   | t_date     | t_datetime          | t_timestamp         | t_year |
+----------+------------+---------------------+---------------------+--------+
| 10:57:50 | 2018-07-29 | 2008-08-08 20:00:00 | 2018-07-29 11:00:12 |   2008 |
+----------+------------+---------------------+---------------------+--------+
1 row in set (0.00 sec)
```

now()函数的使用：

```
mysql> insert into shijian1(t_time, t_date, t_datetime, t_year)values
    -> ('10:57:50','2018-7-29', now() , '2008');
Query OK, 1 row affected (0.00 sec)

mysql> select * from shijian1;
+----------+------------+---------------------+---------------------+--------+
| t_time   | t_date     | t_datetime          | t_timestamp         | t_year |
+----------+------------+---------------------+---------------------+--------+
| 10:57:50 | 2018-07-29 | 2008-08-08 20:00:00 | 2018-07-29 11:00:12 |   2008 |
| 10:57:50 | 2018-07-29 | 2018-07-29 11:02:23 | 2018-07-29 11:02:23 |   2008 |
+----------+------------+---------------------+---------------------+--------+
2 rows in set (0.00 sec)
```

表示当前时间，是mysql中的系统函数，直接用就可以

更新数据，以对比 datetime 类型的数据和 timestamp 类型的数据的区别：

```
mysql> update shijian1 set t_year = 2009 where t_year=2008;
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from shijian1;
+----------+------------+---------------------+---------------------+--------+
| t_time   | t_date     | t_datetime          | t_timestamp         | t_year |
+----------+------------+---------------------+---------------------+--------+
| 10:57:50 | 2018-07-29 | 2008-08-08 20:00:00 | 2018-07-29 11:04:14 |   2009 |
| 10:57:50 | 2018-07-29 | 2018-07-29 11:02:23 | 2018-07-29 11:04:14 |   2009 |
+----------+------------+---------------------+---------------------+--------+
2 rows in set (0.00 sec)
```

这两个时间都自动修改了（重新获得当前时间）

这个没有变化

小细节：timestamp 类型在一个表中只能用于一个字段！

6.5. 字符串型

字符串类型常用的包括：char， varchar， text， enum， set，分述如下：

6.5.1. 定长字符 char 和变长字符 varchar

- 定长字符类型 char：
  适用于存储的字符长度为固定长度的字符，比如中国邮政编码，中国身份证号码，手机号码等。
  设定形式：
  字段名称 char(字符个数)
  其特点是：
  1，存储的字符长度固定，最长可设定为 255 个字符。
  2，如果实际写入的字符不足设定长度，内部会自动用空格填充到设定的长度。
  3，相对 varchar 类型，其存取速度更快。
- 变长字符类型 varchar：
  适用于存储字符长度经常不确定的字符，比如姓名，用户名，标题，内容，等大多数场合的字符。
  设定形式：
  字段名称 varchar(字符个数)
  其特点是：
  1，存储的字符长度是写入的实际长度，但不超过设定的长度。最长可设定为 65532（字节）。
  (1) 注：由于其最长的限制是字节数，因此存储中文和英文的实际字符个数是不同的；
  (2) 英文：一个字符占一个字节；
  (3) 中文（gbk 编码）：一个字符占 2 个字节；
  (4) 中文（utf8 编码）：一个字符占 3 个字节；
  2，如果实际写入的字符不足设定的长度，就按实际的长度存储。
  3，相对于 char 字符串，其存取速度相对更慢。

示例：

定义一个表，演示 char 和 varchar 的使用和区别：

```
定义一个表，演示char和varchar的使用和区别：
create table char_varchar(
    id int auto_increment primary key,
    postcode char(6),
    user_name varchar(6)
);
insert into char_varchar( id, postcode, user_name) values
                        (null, '100110', 'lisi1');

insert into char_varchar( postcode, user_name) values
                        ('120130', 'lisi2');
insert into char_varchar( postcode, user_name) values
                        ('120150', '罗马里奥');

下面给出非常规（非法）的数据情形：
insert into char_varchar( postcode, user_name) values
                        ('120130150', 'lisi3333');
insert into char_varchar( postcode, user_name) values
                        ('120140', 'lisi3333');
    以上两行都报错！

演示不足个数的情况：
insert into char_varchar( id, postcode, user_name) values
                        (null, '110', 'ls4');
    //上一行也可以执行成功！但：
        //postcode字段的值仍然占6个字符
        //而user_name字段只真3个字符
```

思考题：

一个表中有一个字段为 c1，其类型为 char(10)，另有一个字段为 c2，类型为 varchar，问：

c2 最多可以设置多长？

答：

1，一个表中的行也有一个“最大字节长度的限制”，一行最多存储 65532 字节。

2，则此时，c2 最多可以设置 65532-10 = 65522（长度设定）

3，如果 c2 中存储的全是英文字符，就可以存储 65522 个

4，如果存储中文：

gbk：最多 65522/2 = 32761 个

utf8：最多 65522/3 = 21840 个

6.5.2. text 长文本类型

适用于存储“较长的文本内容”，比如文章内容。最长可存储 65535 个字符。

如果还需要存储更长的文本，可以使用 mediumtext（1600 万左右）或 longtext（40 亿左右）。

设定形式：

字段名称 text

text 类型的字段不能设置默认值。
text 类型虽然是字符类型，但不能设置长度！！！
text 类型的数据不存在行中。

```
//演示有关text类型和其他相关类型
create table article(    -- 文章表
    id int auto_increment primary key,
    title varchar(100), -- 标题
    author varchar(20), -- 作者
    content text,
    pub_time datetime, -- 发布时间
    edit_time timestamp -- 更新时间
);
insert into article (title, author, content, pub_time) values
    ('文章标题1','张三','文章内容1','2018-8-8 12:15:30');

insert into article (title, author, content, pub_time) values
    ('文章标题2','张4','文章内容2', now() );
```

6.5.3. enum 和 set 类型

enum 类型和 set 类型都是用于存储“有给定值的可选字符”，比如类似表单中的单选，多选，下拉列表。

- enum 类型（单选类型/枚举类型）：
  enum 类型通常用于存储表单中的“单选项”的值。
  设定形式：
  enum(‘选项值 1’, ‘选项值 2’, ‘选项值 3’, ...)
  这些选项值都对应了相应的“索引值”，类似索引数组的下标，但是从 1 开始的。
  即这些选项的索引值分别为：1， 2， 3， 4， .....
  enum 类型最多可设定 65535 个选项。
  示例：
  create table tab1 (id int, edu(‘大学’, ‘中学’, ‘小学’, ‘其他’));
  insert into tab1 (id, edu) values (1, ‘大学’);
  或：
  insert into tab1 (id, edu) values (1, 2); //表示中学
- set 类型（多选类型）：
  set 类型通常用于存储表单中的“多选项”的值。
  设定形式：
  set(‘选项值 1’, ‘选项值 2’, ‘选项值 3’, ...)
  这些选项值都对应了相应的“索引值”，其索引值从 1 开始，并“依次翻倍”。
  即这些选项的索引值分别为：1， 2， 4， 8， 16， ..... （其实就是 2 的 n 次方）
  enum 类型最多可设定 64 个选项值。
  示例：

create table tab2 (aihao('篮球', '排球', '足球', '中国足球')); #对应索引值为 1,2,4,8
insert into tab2(aihao) values ('篮球');
或：
insert into tab2 (aihao) values ('篮球,排球');
或：
insert into tab2 (aihao) values ('篮球,足球,排球');
或：
insert into tab2 (aihao) values (2); //表示排球(2)
或：
insert into tab2 (aihao) values (3); //表示“篮球,排球”(1+2)
或：
insert into tab2 (aihao) values (7); //表示“篮球,排球,足球”(1+2+4)

基本示例：

```
94
95  enum和set类型：
96  create table user3(
97      id int auto_increment primary key,
98      user_name varchar(20),
99      user_pass char(32),
100     edu enum('小学', '中学','大学'),     -- 单选项数据
101     aihao set('篮球','排球','足球','中国足球') -- 多选选数据
102 );
103 insert into user3 (user_name, user_pass, edu, aihao)
104             values('user1','123','小学','篮球');
105 insert into user3 (user_name, user_pass, edu, aihao)
106             values('user2','123','中学','篮球,排球');
107
108 select * from user3;
```

使用索引值来插入数据：

```
110 使用索引号来插入数据：
111 insert into user3 (user_name, user_pass, edu, aihao)
112             values('user3','123', 3 , 4 );
113
114 insert into user3 (user_name, user_pass, edu, aihao)
115             values('user4','123', 3 , 5 );
116             //5 代表1+4,也就是篮球和足球
117 insert into user3 (user_name, user_pass, edu, aihao)
118             values('user5','123', 2 , 7 );
119             //7 代表1+2+4,也就是篮球和足球
120
```

```
mysql> select * from user3;
+----+-----------+-----------+------+----------------+
| id | user_name | user_pass | edu  | aihao          |
+----+-----------+-----------+------+----------------+
|  1 | user1     | 123       | 小学 | 篮球           |
|  2 | user2     | 123       | 中学 | 篮球,排球      |
|  3 | user3     | 123       | 大学 | 足球           |
|  4 | user4     | 123       | 大学 | 篮球,足球      |
|  5 | user5     | 123       | 中学 | 篮球,排球,足球 |
+----+-----------+-----------+------+----------------+
5 rows in set (0.00 sec)
```

7. 列属性

列属性是指定义或创建一个列的时候，可以给列额外增加的“附加特性”。

形式如下：

create table 表名（列名 列类型 [列属性...]）；

说明：

1，一个列可以有多个列属性；

2，多个列属性空格隔开就行；

列属性包括以下这些：

- null，not null
  - 设定为空，或非空，表明该列数据是否可为空值（null）。
- default
  - 用于设定列默认值（不给值或给空值 null 并 not null，就会自动使用该值）。
  - 使用形式：default 默认值 。
- primary key
  - 用于设定主键。
  - 主键就是一个表中数据的“关键值”，通过该关键值就可以找到该特定的数据行。
  - 一个表的主键值不能重复（相等），比如文章表中的文章编号 id，比如用户表中的用户名。
  - 主键字段必须有值（不能为空）。
  - 一个表只能有一个主键（但一个主键可以是 1 个字段或 2 个以上的字段联合构成）
- auto_increment
  - 用于设定一个整数字段的值是“自增长的”，通常用于一个表中的数据行的编号（比如文章编号）。
  - 默认情况下自增长值从 1 开始。
  - 一个表只能设定一个字段为自增长特性。
- unique key
  - 用于设定“唯一键”的特性。
  - 唯一键表示一个表中的某字段的值是“唯一的”，“不重复的”。
  - 唯一键有点类似 primay key，但其值可以为空（null）。
  - 一个表可以有多个唯一键。
- comment
  - 用于设定字段的说明性内容，类似注释，但又不是注释（属于有效的代码）。
  - 使用形式： comment ‘文字内容’

演示 1：

```
演示null/not null, default, primary key 三个字段属性：
create table shuxing_test1 (
    id int primary key,
    userName varchar(10) not null,
    sex enum('男','女')  default '男' not null,
    f4 float null   -- 其实null可以不写，因为每个字段默认就是null
);
数据测试如下：
insert into shuxing_test1(id, userName) values(1,'user1');
insert into shuxing_test1(id, userName, sex, f4) values(2,'user2','女',10000);
//下面演示错误的数据
1，跟primary 冲突：
insert into shuxing_test1(id, userName, sex, f4) values(1,'user3','男',20000);
2，跟 not null 冲突：
insert into shuxing_test1(id, userName,sex, f4)values(3, null, '男',15000);
insert into shuxing_test1(id,sex, f4)values(3, '男',15000);
```

演示：

创建一个表，并用上以上所有字段属性。字段可包括：id, kecheng, keshi, intro.

```
//综合演示所有字段属性：
创建一个表，并用上以上所有字段属性。字段可包括：id, kecheng, keshi, intro.
create table shuxing_test2(
    id int auto_increment primary key comment '编号值',
    kecheng varchar(20) not null unique key comment '课程名称，不能重复',
    keshi tinyint unsigned default 1 comment '课时',
    intro varchar(1000) comment '课程介绍，应该限制在1000个字符以内为妥'
);
insert into shuxing_test2  values(null, 'PHP', 6, 'web领域最流行的后端语言');
```

联合主键演示：

```
有关primary key 的进一步探讨：
1，主键的另一种设定方式：
create table tab1 (
    id int auto_increment,
    name varchar(10),
    primary key(id)      -- 这就是主键的另一种设定方式！
    );

2，多字段主键的含义及设定：
数据样例（成绩表）：
学员id  科目    成绩
1       mysql   88
2       PHP     90
1       js      77
2       mysql   85
这种情况，就需要设置“多字段主键”，具体做法如下：
create table  chengji (
    学员ID int,
    科目 varchar(20),
    成绩 tinyint unsigned,
    primary key(学员ID,科目)
);
```

昨日回顾：

create table 表名(

字段名 1 类型 [字段附加属性],
字段名 2 类型 [字段附加属性],
....
primary key(xx1, xx2, ...),
unique key (xx1, xx2, ...)
)
charset= utf8/gbk -- 表中存储数据的字符编码
engine=InnoDB/MyIsam/BDB/Memory -- 表类型/存储引擎

类型：
数字类型：
整数：int, tinyint, smallint, mediumint, bigint
小数：float , double, decimal(M, D);
时间类型：
time, date, datetime, year, timestamp,
字符类型：
char: 定长字符，最大可设定 255（字符个数）

varchar: 变长字符，最大可设定为 65532 个字符
text：长文本，不能设定长度，其中最长能存储 65535 个字符
enum：单选型字符，枚举字符，
enum(‘选项 1’,’选项 2’,’选项 3’,....)，他们每个选项对象的索引值为：1,2,3,4,5,6 ,.....
set：多选型字符，
set(‘选项 1’,’选项 2’,’选项 3’,....)，他们每个选项对象的索引值为：1,2,4,8,16 ,.....

属性：

not null 设定为非空数据
default 默认值
primary key 主键，一个表只能设置一个
unique key 唯一键
auto_increment 自增 用于整数，并且一个表只能设置一个
comment ‘说明文字’

8. 实体与实体的关系

8.1. 基本概念

实体（Entity）：

指现实中具体存在的可指称的“某物”。
一个表中的一行数据实际就是指对某物的描述性数据，所以一行数据就是一个实体。
有时实体也指整个表（因为表是由多个实体构成的）。

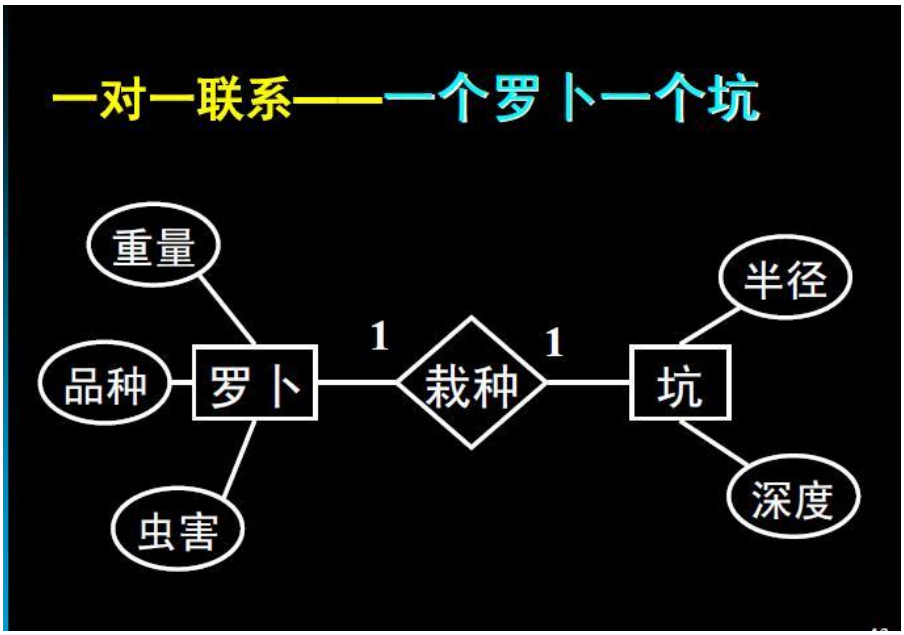
实体间关系（relationship）：

是指不同实体数据之间的关系，很多时候就是指表和表之间的关系。
实体间关系有：一对一关系，一对多关系，多对多关系。

图示如下：

8.2. 一对一关系

表示一个表跟另一个表之间的数据之间一对一的关系。图示如下：



现实案例：

学校表：id，校名， 地址， 校长 id

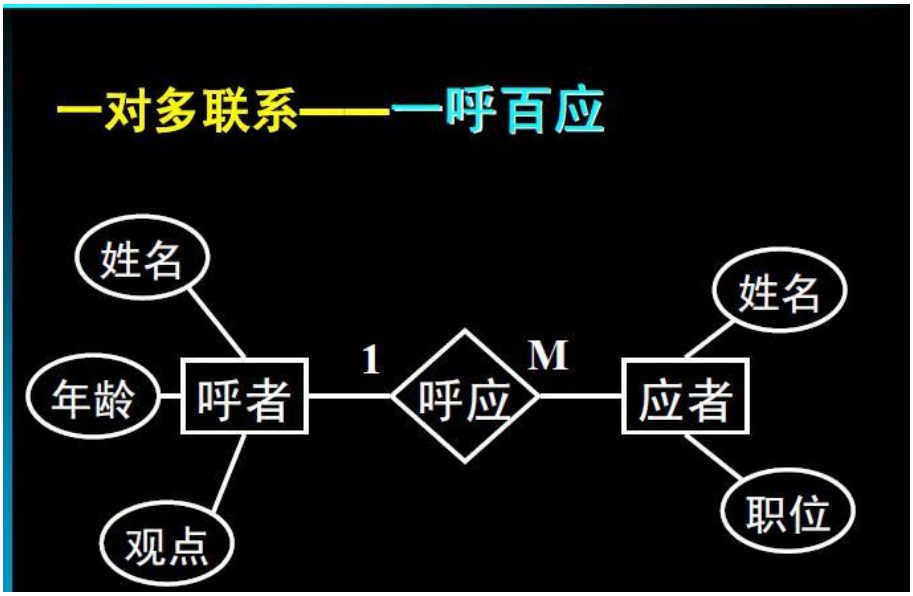
校长表：id，姓名， 年龄， 学历

此时，学校表和校长表就是一对一的关系：

一个学校只能有一个校长，一个校长只能负责一个学校。

8.3. 一对多关系

表示一个表跟另一个表之间的数据之间是一对多的关系。图示如下：



现实案例：

学校表：id，校名， 地址， 校长 id

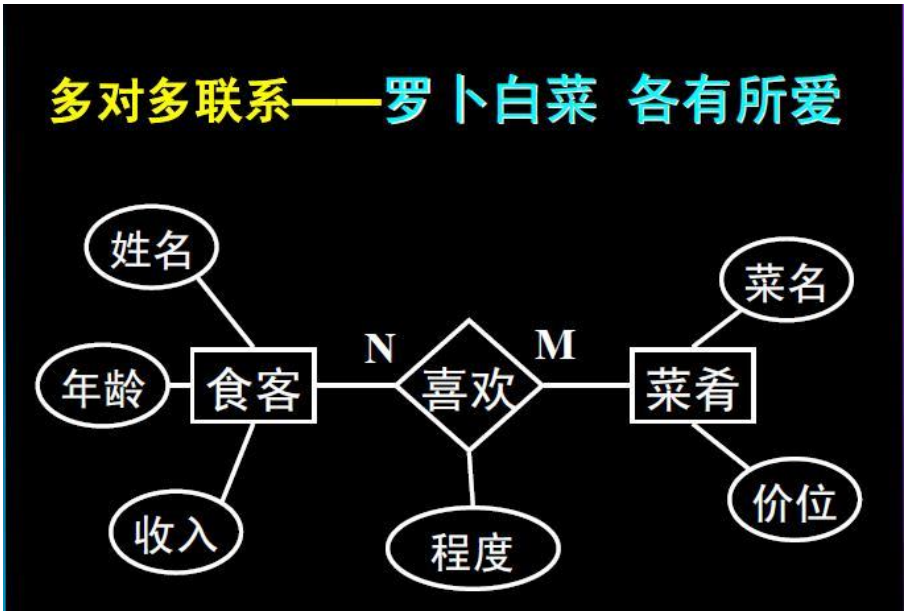
班级表：id，班级名称， 教室号， 所属学校 id

此时，学校表和班级表就是一对多的关系：

一个学校可以有多个班级，一个班级只能属于一个学校。

8.4. 多对多关系

表示一个表跟另一个表之间的数据之间是多对多的关系。图示如下：



现实案例：

课程表：id，课程名称， 课时数， 学分数

学生表：id，姓名， 性别， 年龄， 所属班级 id

此时，课程表和班级表就是多对多的关系：

一个课程可以被多个学生学习，

一个学生也可以学多个课程。

这种情况，通常还需要建立一个“中间表”，以记录所有学生各自选修了哪些课程，如下所示：

学生 id	课程 id	选修时间
1	1	2017-8-9
1	3	2017-8-9
1	4	2017-9-1
2	1	2017-8-4
2	2	2017-8-4
2	3	2017-9-1
3	3	2018-6-9
...	...	...

》》》day3

9. 高级查询

9.1. 高级查询语法概述

一个查询语句的完整形式如下所示：

```
select  子句
    [from  子句]
    [where  子句]
    [group by 子句]
    [having 子句]
    [order by 子句]
    [limit  子句]
    ;
```

可见，select 语句还是比较复杂的——其实是 mysql 中最复杂的语句。

总体说明：

1，以上中括号中的任一项都可以省略，但如果不省略，就应该按该顺序出现。

2，通常，from 后的子句都需要有 from 子句， having 子句需要有 group by 子句。

3，这些子句的“执行顺序”，也是按此顺序进行的。

9.2. 查询结果数据及 select 选项

9.2.1. 查询“固定数据”

例：

```
select  1;
select  2,  ‘abc’;
select  3,  now();
```

9.2.2. select 中可以进行计算

例：

```
select  1 + 2;
```

```
select  3+4*5,  6 + round(6.7) ;      #其中 round( )为系统函数
```

9.2.3. 查询出的数据字段可以使用别名

例：

```
select  1  as  d1,  2+3  as  d2;
select  user_name  as un,  user_pass  as pwd  from  users;
```

9.2.4. 使用 distinct 消除查询结果重复行

重复行的含义：

两行（或两行以上）的数据完全一样。

语法形式：

```
select  disctinct  字段 1，  字段 2，  ...  from  表名；
```

9.3. where 子句

语法形式：

```
select  ....  from  表名  where  查询条件；
```

说明：

1，查询条件，类似 php 语言中的判断条件，也就是说，where 相当于 if。

2，查询条件的作用是：针对 from 子句的表“进行一行一行筛选”，以筛选出符合条件的行。

3，查询条件中，可以使用很多的运算符，包括：算术运算符，比较运算符，逻辑运算符，等等。

示例：

```
where  id > 10;   //比较运算符
where  age - 18 >= 0;  //算术运算符，比较运算符
where  id < 20  and  age >= 18;     //比较运算符，逻辑运算符
where  year  %  4 = 0  and  year  %  100  !=  0  ||  year  % 400  =  0;
                                    //算术运算符，比较运算符，逻辑运算符
```

9.4. mysql 运算符

9.4.1. 算术运算符

```
+   -   *   /   %
```

9.4.2. 比较运算符：

相等： =

不相等： <> 或 !=

大于： >

大于等于： >=
小于： <
小于等于： <=

9.4.3. 逻辑运算符：

逻辑与： && 或 and
逻辑或： || 或 or
逻辑非： ! 或 not

导入范例数据表（电子商城表）：

```
mysql> source H:\itcast\mysql\课堂用例：连接查询.sql
Query OK, 0 rows affected (0.00 sec)
```

9.4.4. 其他特殊运算符

like 模糊查找运算符：

用于判断某个字符型字段的值是否包含给定的字符。

语法形式：

xxx 字段 like ‘%关键字%’

其中：%表示“任意个数的任意字符”。

还可以使用“_”（下杠），表示“任意一个字符”。

```
where name like ‘罗%’        //找出 name 的第一个字为“罗”的所有
                             //可以找出：”罗成”,“罗永浩”，“罗纳尔多”，“罗”
                             //但找不出“c 罗纳尔多”这个
where name like ‘罗_’        //可以找出：”罗成”, “罗兰”,
                             //但找不出“c 罗”，“罗永浩”
```

极端情况：

```
where name like “罗”         //其实它只是相当于： name = ‘罗’
```

如果不使用“%”或“_”，则 like 相当于等于（=）。比如：

```
xxx 字段 like ‘关键字’
```

相当于：

```
xxx 字段 = ‘关键字’
```

between 范围限定运算符：

用于判断某个字段的值是否在给定的两个数据范围之间。

语法形式：

xxx 字段 between 值 1 and 值 2

其含义相当于： xxx 字段 >= 值 1 and xxx 字段 <= 值 2

in 运算符：

用于判断某个字段的值是否在给出的若干个“可选值”范围。

语法形式：

xxx 字段 in （值 1， 值 2， ..... ）

其含义是：该字段的值等于所列出的任意一个值，就算满足条件，比如：

籍贯 in (‘北京’， ‘山东’， ‘河北’， ‘江西’); //则某人籍贯为上述 4 个之一就 ok。

is 运算符：

用于判断一个字段中的是“是否存在”（即有没有），只有两个写法，如下所示：

```
where content is null;          //不能写成： content = null;
where content is not null;      //不能写成： content != null;
```

9.5. group by 子句

语法形式：

group by 字段 1， 字段 2， .... ；

含义：

表示对所取得的数据，以所给定的字段来进行分组。

最后的结果就是将数据分成了若干组，每组作为一个“整体”成为一行数据。

特别注意：

分组之后，只有“组信息”——一行就是一组

示例：

对于如下原始数据：

商品信息表(product)

pro_id 商品ID	pro_name 商品名称	protype_id 商品类别ID	price 价格	pinpai 品牌	chandi 产地
1	康佳（KONKA）42英寸全高清液晶电视	1	1999	康佳	深圳
2	索尼（SONY）4G手机（黑色）	2	3238	索尼	深圳
3	海信（Hisense）55英寸智能电视	1	4199	海信	青岛
4	联想（Lenovo）14.0英寸笔记本电脑	3	5499	联想	北京
5	索尼（SONY）13.3英寸触控超极本	3	11499	索尼	深圳
11	索尼（SONY）60英寸全高清液晶电视	1	6999	索尼	北京
12	联想（Lenovo）14.0英寸笔记本电脑	3	2999	联想	北京
13	联想 双卡双待3G手机	2	988	联想	北京
15	惠普（HP）黑白激光打印机	3	1169	惠普	天津

对其按“品牌”进行分组：

```
SELECT * FROM `product` group by pinpai
```

结果为：

pro_id	pro_name	protype_id	price	pinpai	chandi
1	康佳（KONKA）42英寸全高清液晶电视	1	1999.00	康佳	深圳
15	惠普（HP）黑白激光打印机	3	1169.00	惠普	天津
3	海信（Hisense）55英寸智能电视	1	4199.00	海信	青岛
2	索尼（SONY）4G手机（黑色）	2	3238.00	索尼	深圳
4	联想（Lenovo）14.0英寸笔记本电脑	3	5499.00	联想	北京

特别注意：

分组查询的结果，要理解为，将“若干行原始数据”，分成了若干组，结果是每组为一行数据。

即：一行数据就代表“一组”这个集合概念，而不再是单个概念。

因此：一行中出现的信息，应该是“组的信息”，而不是“个体信息”。

于是，对于分组查询（group by），select 中出现的信息，通常就只有两种情况的信息了：

1，分组本身的字段信息；

2，一组的综合统计信息，主要包括：

(1) 计数值： count(字段)， 表示求出一组中原始数据的行数；

(2) 最大值： max(字段)，表示求出一组中该字段的最大值；

(3) 最小值： min(字段)，表示求出一组中该字段的最小值；

(4) 平均值： avg(字段)，表示求出一组中该字段的平均值；

(5) 总和值： sum(字段)，表示求出一组中该字段的累加和；

以上 5 个函数，也称为“聚合函数”！

示例：

```
41  示例1：
42  查询出各个品牌的产品的平均价。
43  select pinpai, avg(price)  from product  group by pinpai;
44
45  示例2：
46  查询出各个产地的产品数量、平均价，最高价，最低价。
47  SELECT chandi, count(*) as 数量, avg(price) as 平均价, max(price) as 最高价,
48      min(price)  FROM `product` group by chandi
49
50  示例3：
51  查询出产品表中的产品总数。
52  select count(*) as 产品总数量 from product;
53
54  示例4：
55  查询出产品表中联想品牌的产品总数。
56  select count(*) as 总数 from product where pinpai = '联想'
```

多条件分组：

将 product 表中的所有商品以品牌和产地进行分组，并求出每一组的数量

select pinpai, chandi, count(*) as 数量 from product group by pinpai, chandi;

9.6. having 子句

语法形式：

having 筛选条件

含义：

having 的含义跟 where 的含义一样，但 having 是只用于对 group by 分组的结果进行的条件筛选。

即：having 其实是相当于分组之后“有若干行数据”，然后对这些行再筛选。

示例：

查询出品牌平均价超过 5000 的所有品牌的平均价，最高价，以及产品的数量。

```
mysql> select pinpai,avg(price) as 平均价, max(price) as 最高价,
    -> count(*) as 数量 from product group by pinpai having avg(price) > 5000;
+--------+-------------+----------+------+
| pinpai | 平均价      | 最高价   | 数量 |
+--------+-------------+----------+------+
| 索尼   | 7245.333333 | 11499.00 |    3 |
+--------+-------------+----------+------+
1 row in set (0.00 sec)
```

再来一个：

```
--查询出价格超过3000的所有产品的品牌平均价超过5000所有品牌的平均价，最高价，以及产品的数量
select pinpai, avg(price) as 平均价 from product where price > 3000 group by pinpai having 平均价>5000;
```

9.7. order by 子句

语法形式：

order by 字段 1 [asc 或 desc]， 字段 2 [asc 或 desc]， ....

含义：

对前面所取得的数据按给定的字段进行排序。

排序方式有： 正序 asc， 倒序 desc，如果省略不写，就是 asc

示例：

```
演示order by的使用：
示例1： 对所有产品按价格从高到低进行排序；
select * from product order by price desc;

--对所有品牌的平均价按从高到低的顺序进行排序，并列出品牌名和平均价。
select pinpai, avg(price) as 平均价 from product group by pinpai order by 平均价 desc ;
```

9.8. limit 子句

语法形式：

limit 起始行号，行数

说明：

1，limit 表示对前面所取得的数据再进行数量上的筛选：取得从某行开始的多少行。

2，行号就是前面所取得数据的“自然顺序号”，从 0 开始算起——注意不是 id，或任何其他实际数据。

3，起始行号可以省略，此时 limit 后只用一个数字，表示从第 0 行开始取出多少行。

4，limit 子句通常用在“翻页”功能上，用于找出“第 n 页”的数据，其公式为：

limit (n - 1) * pageSize, pageSize; 其中 pageSize 表示每页显示的条数。

示例 1：

取出商品表中价格最高的 3 个商品，并按倒序排列出来。

```
select * from product order by price desc limit 0,3;
```

应用：

limit 子句常常用于网页的“翻页功能”。

假设总的数据行数为 9，每页显示 2 行（条），则：

查看第 1 页： select * from product limit 0, 2;

查看第 2 页： select * from product limit 2, 2;

查看第 3 页： select * from product limit 4, 2;

.....

查看第 n 页： select * from product limit (n-1)*2, 2;

10. 高级插入

10.1. 同时插入多行记录

语句形式：

insert into 表名(字段 1，字段 2，...) values (值 1，值 2，...), (值 1， 值 2，...), ...;

```
insert into user2 (id, name, sex, age)values(4,'张5','男', 18),(5,'张6','女', 19)
```

10.2. 插入查询的结果数据

含义：

就是将一个 select 语句的查询结果，插入到某个表中！

语句形式：

insert into 表名(字段 1，字段 2，...) select (xx1, xx2, ...) ... ;

要求：

1， 插入语句的字段个数，跟 select 语句的字段个数相等；

2， 插入语句的字段类型，跟 select 语句的字段类型相符；

```
mysql> insert into user3 (user_name, user_pass) select name, '456' as pass from user2;
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from user3;
+----+-----------+-----------+------+-------------------------+
| id | user_name | user_pass | edu  | aihao                   |
+----+-----------+-----------+------+-------------------------+
|  1 | user1     | 123       | 小学 | 篮球                    |
|  2 | user2     | 123       | 中学 | 篮球,排球               |
|  3 | user3     | 123       | 大学 | 足球                    |
|  4 | user4     | 123       | 大学 | 篮球,足球               |
|  5 | user5     | 123       | 中学 | 篮球,足球               |
|  6 | user4     | 123       | 大学 | 篮球,排球,足球,中国足球 |
|  7 | user4     | 123       | 大学 | 排球,足球,中国足球      |
|  8 | 张三      | 456       | NULL | NULL                    |
|  9 | 李四      | 456       | NULL | NULL                    |
| 10 | 李5       | 456       | NULL | NULL                    |
| 11 | 张5       | 456       | NULL | NULL                    |
| 12 | 张6       | 456       | NULL | NULL                    |
+----+-----------+-----------+------+-------------------------+
12 rows in set (0.00 sec)
```

这是新插入的数据

10.3. set 语法插入数据

语句形式：

insert into 表名 set 字段 1=值 1， 字段 2=值 2，.... ；

```
93  insert into user3 set user_name='user6', user_pass='123', edu=2, aihao=3 ;
94
```

10.4. 蠕虫复制

所谓蠕虫复制，就是针对一个表的数据，进行快速的复制并插入到所需要的表中，以期在短时间内具备“大量数据”，以用于测试或其他特殊场合，比如：

1，将一个表的大量数据，复制到另一个表中；

```
mysql> CREATE TABLE `user3a` (
    ->   `id` int(11) NOT NULL AUTO_INCREMENT,
    ->   `user_name` varchar(20) DEFAULT NULL,
    ->   `user_pass` char(32) DEFAULT NULL,
    ->   `edu` enum('小学','中学','大学') DEFAULT NULL,
    ->   PRIMARY KEY (`id`)
    -> ) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=utf8
    -> ;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into user3a (user_name, user_pass,edu) select user_name, user_pass,edu from user3;
Query OK, 13 rows affected (0.00 sec)
Records: 13  Duplicates: 0  Warnings: 0

mysql> select * from user3a;
+----+-----------+-----------+------+
| id | user_name | user_pass | edu  |
+----+-----------+-----------+------+
| 16 | user1     | 123       | 小学 |
| 17 | user2     | 123       | 中学 |
| 18 | user3     | 123       | 大学 |
| 19 | user4     | 123       | 大学 |
| 20 | user5     | 123       | 中学 |
| 21 | user4     | 123       | 大学 |
| 22 | user4     | 123       | 大学 |
| 23 | 张三      | 456       | NULL |
| 24 | 李四      | 456       | NULL |
| 25 | 李5       | 456       | NULL |
| 26 | 张5       | 456       | NULL |
| 27 | 张6       | 456       | NULL |
| 28 | user6     | 123       | 中学 |
+----+-----------+-----------+------+
13 rows in set (0.00 sec)
```

刚刚创建的表

执行一个插入语句

结果：user3中的数据，都插入到user3a中了。

2，将一个表的数据复制到本身表中以产生大量数据；

```
mysql> insert into user3a (user_name, user_pass,edu) select user_name, user_pass,edu from user3a;
Query OK, 52 rows affected (0.00 sec)
Records: 52  Duplicates: 0  Warnings: 0

mysql> select * from user3a;
+-----+-----------+-----------+------+
| id  | user_name | user_pass | edu  |
+-----+-----------+-----------+------+
```

10.5. 插入时主键冲突的解决办法

所谓主键冲突是指，当插入一条记录的时候，如果插入的记录的主键值，在现有的数据中已经存在，则此时，因为主键不能重复，因此就产生了“主键冲突”。

主键冲突的演示：

现有数据：

```
mysql> select * from shuxing_test1;
+----+----------+-----+-------+
| id | userName | sex | f4    |
+----+----------+-----+-------+
|  1 | user1    | 男  |  NULL |
|  2 | user2    | 女  | 10000 |
+----+----------+-----+-------+
2 rows in set (0.00 sec)
```

插入一个数据（其主键已经存在的情形）：

```
mysql> insert into shuxing_test1(id, userName, sex, f4)values(2, user3 , 男 ,2000);
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

那如果出现主键冲突，该怎么办呢？

- 办法 1：忽略

  ——终止插入，数据不改变，语句不报错。

  其语法为：

  insert **ignore** into 表名 (字段....) values (值.....);

```
mysql> insert ignore into shuxing_test1(id, userName, sex, f4)values(2,'user3','男',2000);
Query OK, 0 rows affected (0.00 sec)
```

- 办法 2：替换

  ——删除原纪录，插入新纪录。

  其语法为：

  replace into 表名 (字段....) values (值.....);

  说明：此 replace 的用法跟 insert 一样，也可以插入新纪录，只是如果新纪录出现主键冲突，就会删除原纪录后，再插入该新纪录。

```
mysql> replace into shuxing_test1(id, userName, sex, f4)values(2,'user3','男',2000);
Query OK, 2 rows affected (0.00 sec)

mysql> select * from shuxing_test1;
+----+----------+-----+------+
| id | userName | sex | f4   |
+----+----------+-----+------+
|  1 | user1    | 男  | NULL |
|  2 | user3    | 男  | 2000 |
+----+----------+-----+------+
2 rows in set (0.00 sec)
```

被替换（覆盖）后的数据，id(主键)值不变

- 办法 3：更新

  ——设置为去更新原有数据（而并不插入）。

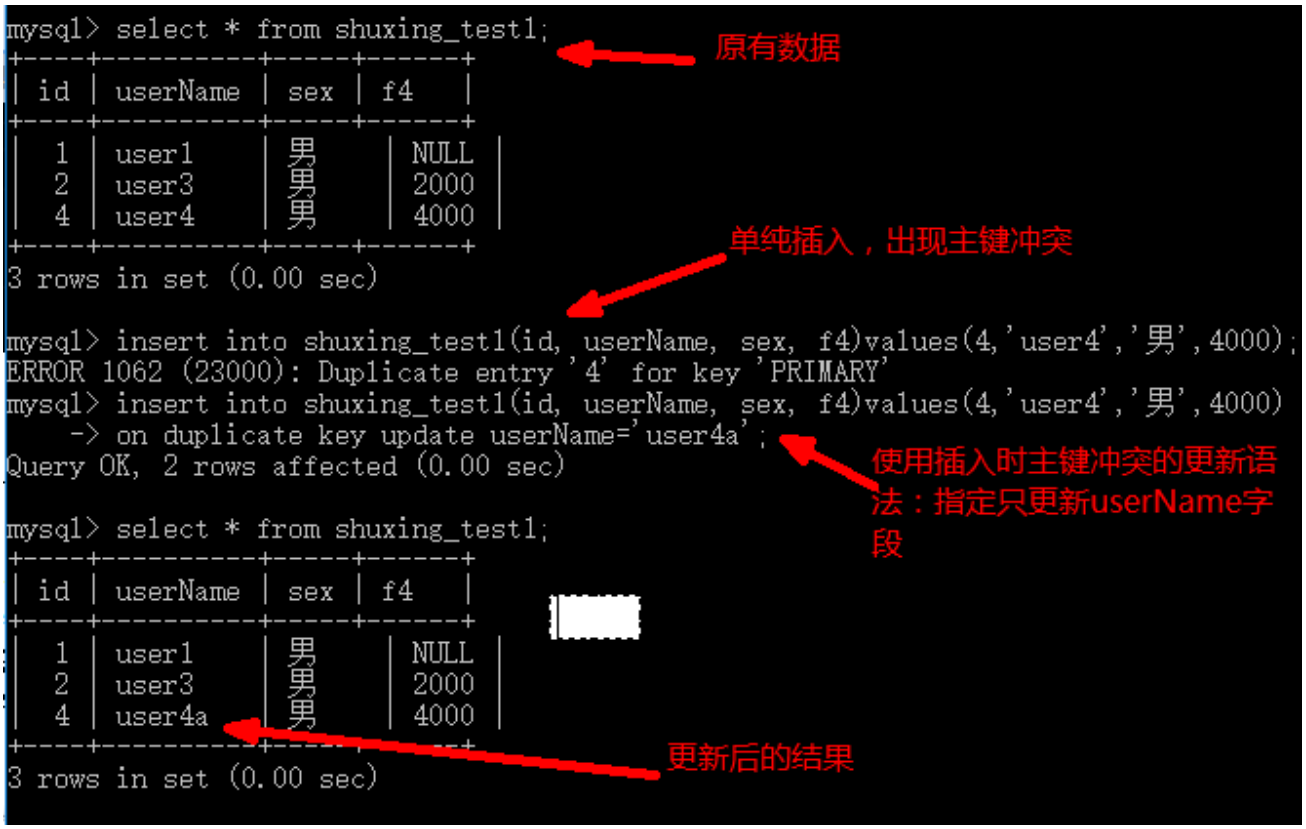
  语法为：

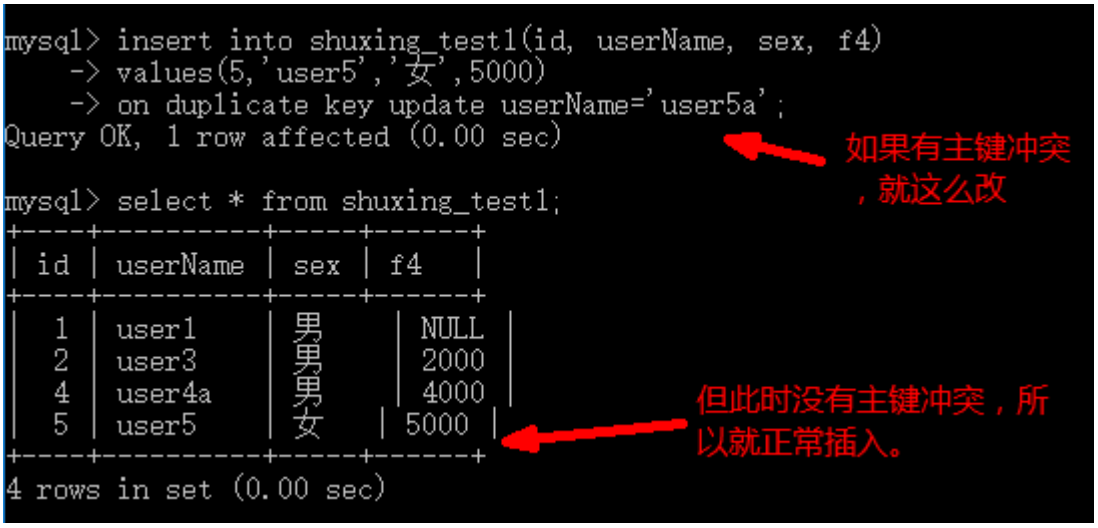
  insert into 表名 (字段....) values (值.....) on duplicate key update XX 字段=新的值;

  更新类似“替换”（replace），区别是：

  替换：是将新的数据完整覆盖旧的数据

  更新：可以预先设定需要覆盖的旧数据





11. 高级删除

11.1. 按指定顺序删除指定数量的数据

语法形式：

delete from 表名 where ... [order by 字段名, ...] [limit 数量 n]；

说明：

1，order by 用于设定删除数据时的删除顺序，跟 select 语句中的 order by 子句道理一样。

2，limit 用于设定删除数据时要删除的行数，即删除的数据可能少于条件筛选出来的数据。

```
mysql> delete from user3a where edu is null order by user_name limit 17;
Query OK, 17 rows affected (0.00 sec)
```

11.2. truncate 清空

语法形式：

truncate 表名；

说明：

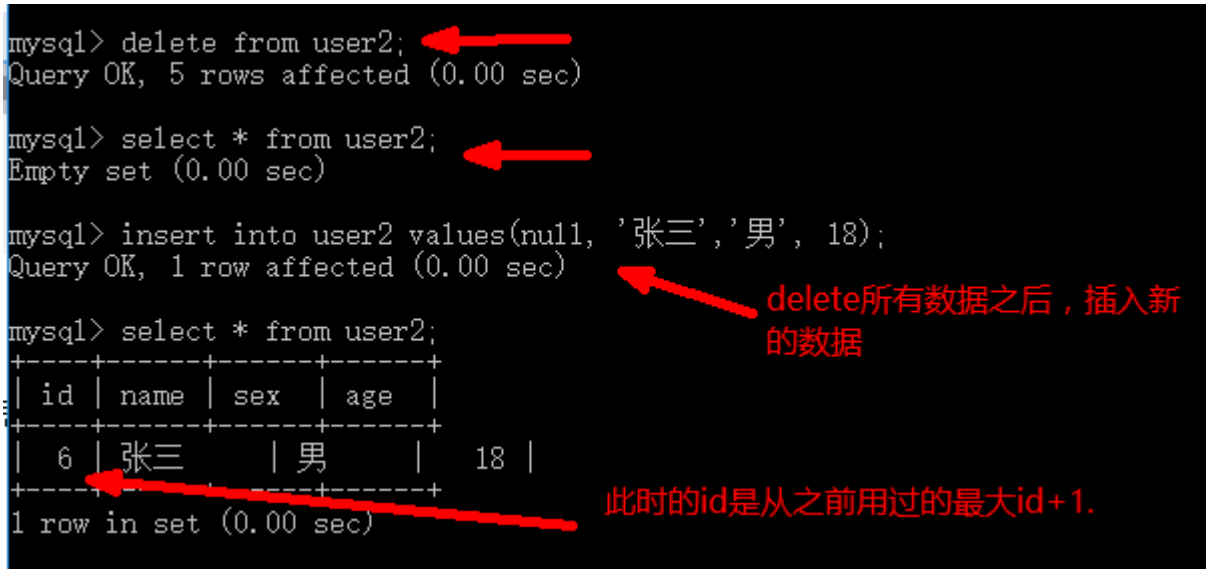
表示清空指定表中的所有数据并将表恢复到“初始状态”（就类似刚刚创建一样）。

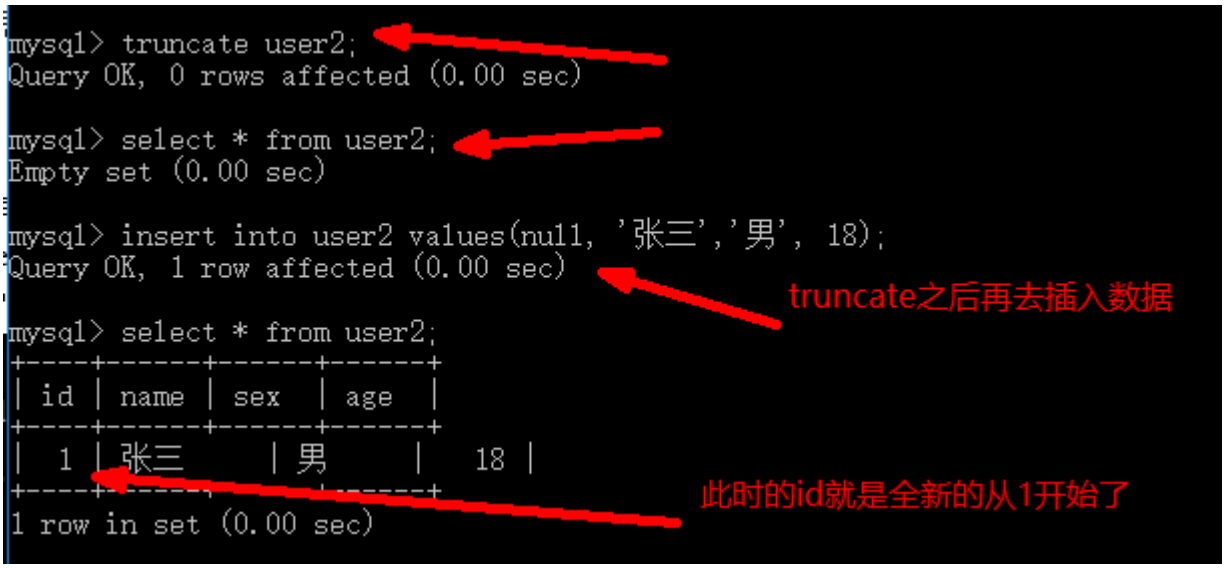
对比：

无条件删除： delete from 表名；

结果：删除了指定表中的所有数据——但表仍然会被纪录为“已使用过”。

差别：主要是对于”auto_increment”的字段，会保留使用过的最大值，而 trancate 后的表，自增长的序号会完全重新开始（就像新表一样）。





12. 高级更新

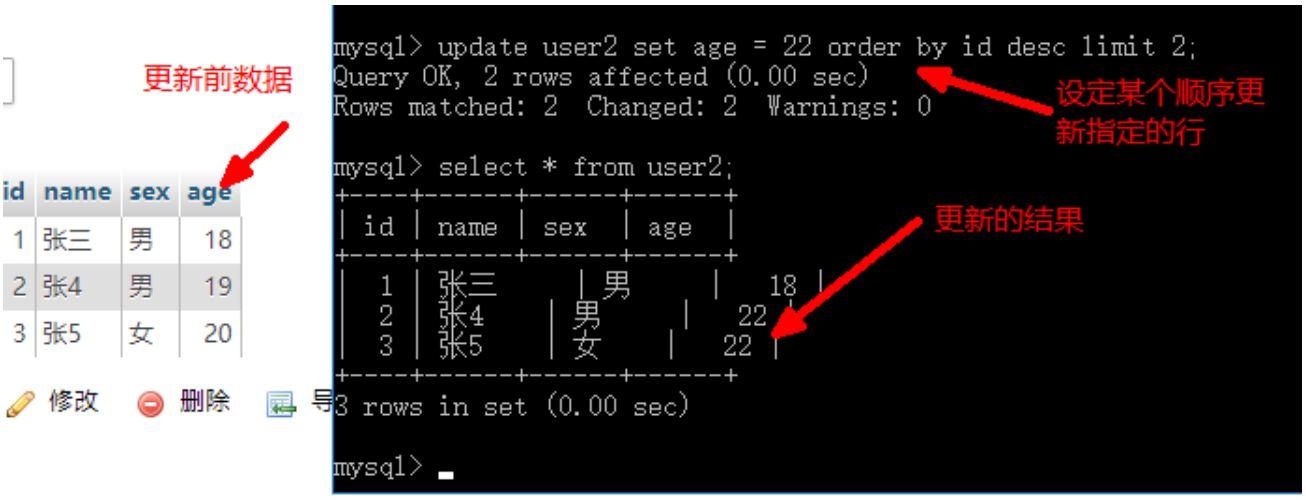
语法形式：

update 表名 set 字段名 1=字段值 1， ... where ... [order by 字段名, ...] [limit 数量 n]；

说明：

1，order by 用于设定更新数据时的更新顺序，跟 select 语句中的 order by 子句道理一样。

2，limit 用于设定更新数据时要更新的行数，即更新的数据量可能少于条件筛选出来的数据量。



昨日回顾

mysql 第 1 天：

系统性操作：
安装系统
登录：
cmd，登录成功，立即设置连接字符:
set names gbk;
退出：
建库：
create database 数据库名 charset utf8;
建表：
建表的核心是：设定若干字段

基本增删改查：
insert into 表名(f1, f2, f3)values(null, 'v1', 18);
$sql = "insert into 表名(f1, f2, f3)values(null, 'v1', 18);";
mysqli_query($link, $sql);
delete from 表名 where id=5;
update ...
select ....

mysql 第 2 天：
数据类型：
数字类型：
整数：int， tinyint
小数：浮动小数，定点小数
字符类型：
char:
varchar:
text:
enum:枚举，单选类型
set：多选类型
时间类型：
time，date， datetime，timestamp，
字段属性：
not null
primary key
auto_increment
unique key
default xx;
comment '说明文字'

mysql 第 3 天：
高级查询：
select xxx
from xxx
where ....

group by ...
having ...
order by ..
limit ...

高级插入：

插入多行

将查询结果插入到某个表中：

insert into xxx（字段列表） select 字段列表 from ...;

主键冲突：

1，直接忽略：

insert ignore into 表名.....

2，替换原数据：

replace into .....

3, 预先设定更新数据：

inser into ..... on duplicate key update

set 某字段=某值, 某字段=某值...

高级删除：

truncate

高级修改：

这两者都只是可以额外增加如下特性：

按什么顺序区删除或修改，并设定可以修改（/删除）多少条

》》》day4

13. 联合（union）查询

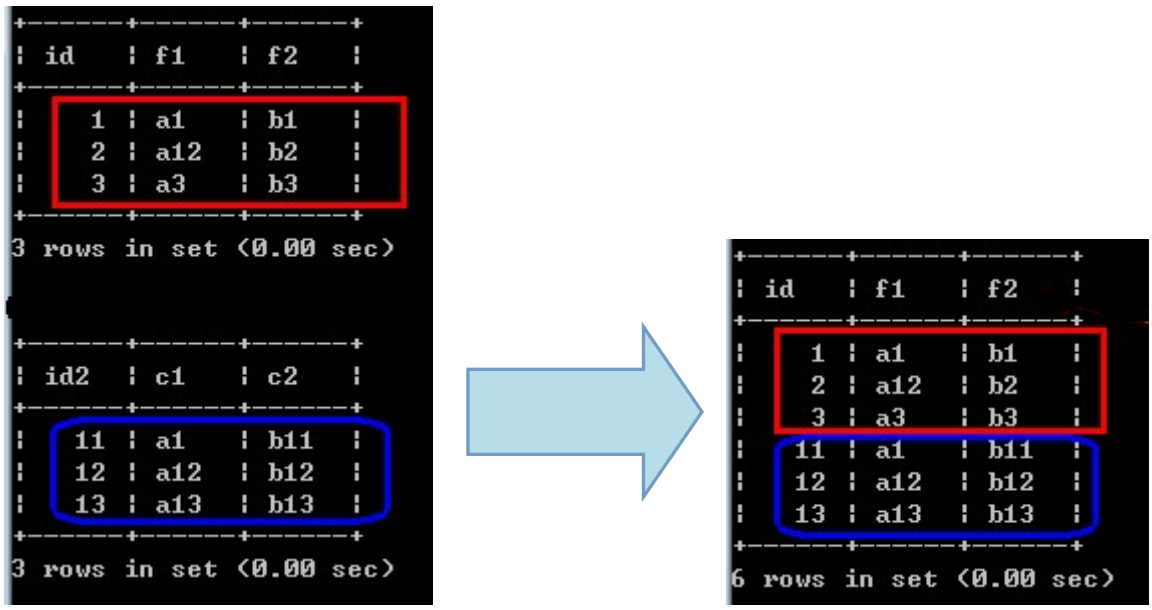
13.1. 联合查询概念

含义：

联合查询是指将 2 个或 2 个以上的字段数量相同的查询结果，“纵向堆叠”后合并为一个结果。

图示如下：

```
select id, f1, f2 from join1
union
select id2, c1, c2 from join2;
```



13.2. 联合查询语法

语法形式：

select 查询 1
union [all 或 distinct]
select 查询 2
union [all 或 distinct]
select 查询 3
.....
[order by 字段 [asc 或 desc]]
[limit 起始行号，数量] ；

说明：

1， 所有单个查询结果应该具有相等的列数。
2， 所有单个查询的列类型应该具有一致性（即每个查询的第 n 列的数据类型一致）。
3， 单个查询的列名可以不同，但最终的列名是第一个查询的列名（可以使用别名）。
4， union 可以带 all 或 distinct 参数，如果省略就是 distinct，即默认已经消除重复行了。
5， 最后的 order by 或 limit 是对整个联合之后的结果数据进行排序或数量限定。
6， order by 子句中的排序字段应该使用第一个查询中的字段名，如果有别名就必须使用别名。
7， 可见，假设：
查询 1 有 n1 行，m 列；
查询 2 有 n2 行，m 列；
则两个表“联合”之后的结果，有最多 n1+n2 行，m 列。

示例：

```
mysql> select * from join1
    -> union
    -> select * from join2;
+------+------+------+
| id   | f1   | f2   |
+------+------+------+
|    1 | a1   | b1   |
|    2 | a12  | b2   |
|    3 | a3   | b3   |
|   11 | a1   | b11  |
|   12 | a12  | b12  |
|   13 | a13  | b13  |
+------+------+------+
6 rows in set (0.00 sec)
```

```
mysql> select id, f2 from join1
    -> union
    -> select id2,c2 from join2;
+------+------+
| id   | f2   |
+------+------+
|    1 | b1   |
|    2 | b2   |
|    3 | b3   |
|   11 | b11  |
|   12 | b12  |
|   13 | b13  |
+------+------+
6 rows in set (0.00 sec)
```

```
15  -- 演示order 和limit的使用:
16  select id, f2 from join1
17  union
18  select id2,c2 from join2
19  order by id desc
20  limit 0,4;
```

```
mysql> select id, f2 from join1
    -> union
    -> select id2,c2 from join2
    -> order by id desc
    -> limit 0,4;
+------+------+
| id   | f2   |
+------+------+
|   13 | b13  |
|   12 | b12  |
|   11 | b11  |
|    3 | b3   |
+------+------+
4 rows in set (0.00 sec)
```

```
mysql> -- 演示"不消除重复行"的情形:
mysql>  select f1 from join1
    -> union all
    -> select c1 from join2;
+------+
| f1   |
+------+
| a1   |
| a12  |
| a3   |
| a1   |
| a12  |
| a13  |
+------+
6 rows in set (0.00 sec)
```


14. 连接（join）查询

连接（join）查询是将两个查询的结果以“横向对接”的方式合并起来的结果。
对比：联合查询 是将两个查询的结果以“纵向堆叠”的方式合并起来的结果。

14.1. 连接查询概述

连接查询，是将两个查询(或表)的每一行，以“两两横向对接”的方式，所得到的所有行的结果。
即一个表中的某行，跟另一个表中的某行，进行“横向对接”，而得到一个新行。
如下图所示：

table1

id	f1	f2
1	a1	b1
2	a12	b2
3	a3	b3

连接就是左边的一行和右边的一行"对接"起来

table2

id2	c1	c2
11	a1	b11
12	a12	b12
13	a13	b13

连接查询（join关键字）的基本含义是：将两个表以“并排”的方式，连接起来，成为“更多字段”的一个新表。当然，根据连接的不同方式，得到的新表的行数会有不同。如图所示。

则他们对接（连接）之后的结果类似这样：

id	f1	f2	id2	c1	c2
1	a1	b1	11	a1	b11
2	a12	b2	11	a1	b11
3	a3	b3	11	a1	b11
1	a1	b1	12	a12	b12
2	a12	b2	12	a12	b12
3	a3	b3	12	a12	b12
1	a1	b1	13	a13	b13
2	a12	b2	13	a13	b13
3	a3	b3	13	a13	b13

9 rows in set (0.00 sec)

这就是完全连接之后的结果，可见：先左边的所有行跟右边 第1行依次连接，然后左边的所有行再跟右边的第2行依次连接，依次类推.....

可见，假设：

表 1 有 n1 行，m1 列；

表 2 有 n2 行，m2 列；

则表 1 和表 2“连接”之后，就会有：

n1*n2 行；

m1+m2 列。

连接查询基本形式如下：

select ... from 表 1 [连接方式] join 表 2 [on 连接条件] where ... ;

可见，连接查询只是作为 from 子句的“数据源”。

或者说，连接查询是扩大了数据源，从原来的一个表作为数据源，扩大为多个表作为数据源。

连接查询包括以下这些不同形式：

交叉连接，内连接，外连接（分：左外连接，右外连接）。

14.2. 交叉连接（cross join）

语法形式：

from 表 1 [cross] join 表 2

说明：

1，交叉连接其实可以认为是连接查询的“完全版本”，即所有行都无条件地都连接起来了。

2，关键字“cross”可以省略；

3，交叉连接又称为“笛卡尔积”，通常应用价值不大。

4，交叉连接还有一种写法： from 表 1， 表 2；

演示：

```
mysql> select * from join1 cross join join2;
+------+------+------+------+------+------+
| id   | f1   | f2   | id2  | c1   | c2   |
+------+------+------+------+------+------+
|    1 | a1   | b1   |   11 | a1   | b11  |
|    2 | a12  | b2   |   11 | a1   | b11  |
|    3 | a3   | b3   |   11 | a1   | b11  |
|    1 | a1   | b1   |   12 | a12  | b12  |
|    2 | a12  | b2   |   12 | a12  | b12  |
|    3 | a3   | b3   |   12 | a12  | b12  |
|    1 | a1   | b1   |   13 | a13  | b13  |
|    2 | a12  | b2   |   13 | a13  | b13  |
|    3 | a3   | b3   |   13 | a13  | b13  |
+------+------+------+------+------+------+
9 rows in set (0.00 sec)
```

```
mysql> select * from join1, join2;
+------+------+------+------+------+------+
| id   | f1   | f2   | id2  | c1   | c2   |
+------+------+------+------+------+------+
|    1 | a1   | b1   |   11 | a1   | b11  |
|    2 | a12  | b2   |   11 | a1   | b11  |
|    3 | a3   | b3   |   11 | a1   | b11  |
|    1 | a1   | b1   |   12 | a12  | b12  |
|    2 | a12  | b2   |   12 | a12  | b12  |
|    3 | a3   | b3   |   12 | a12  | b12  |
|    1 | a1   | b1   |   13 | a13  | b13  |
|    2 | a12  | b2   |   13 | a13  | b13  |
|    3 | a3   | b3   |   13 | a13  | b13  |
+------+------+------+------+------+------+
9 rows in set (0.00 sec)
```

14.3. 内连接（inner join）

语法形式：

from 表1 [inner] join 表2 on 连接条件

说明：

1，内连接其实是交叉连接的基础上，再通过 on 条件而筛选出来的部分数据。

2，关键字“inner”可以省略，但建议写上。

3，内连接是应用最广泛的一种连接查询，其本质是根据条件筛选出“有意义的数据”。

演示：

找出出所有商品及其所属类别。

```
-- 内连接查询:
SELECT * FROM `product` inner join product_type on product.protype_id = product_type.protype_id;
-- 也可以改写为:
SELECT * FROM product as p inner join product_type as t on p.protype_id = t.protype_id;
```

pro_id	pro_name	protype_id	price	pinpai	chandi	protype_id	protype_name
1	康佳（KONKA）42英寸全高清液晶电视	1	1999.00	康佳	深圳	1	家用电器
2	索尼（SONY）4G手机（黑色）	2	3238.00	索尼	深圳	2	手机数码
3	海信（Hisense）55英寸智能电视	1	4199.00	海信	青岛	1	家用电器
4	联想（Lenovo）14.0英寸笔记本电脑	3	5499.00	联想	北京	3	电脑办公
5	索尼（SONY）13.3英寸触控超极本	3	11499.00	索尼	天津	3	电脑办公
11	索尼（SONY）60英寸全高清液晶电视	1	6999.00	索尼	北京	1	家用电器
12	联想（Lenovo）14.0英寸笔记本电脑	3	2999.00	联想	北京	3	电脑办公
13	联想 双卡双待3G手机	2	988.00	联想	北京	2	手机数码
15	惠普（HP）黑白激光打印机	3	1169.00	惠普	天津	3	电脑办公

课堂练习：

找出所有价格大于 5000 的家用电器的商品的完整信息（含所属类别）；

```
SELECT * FROM product as p inner join product_type as t on p.protype_id = t.protype_id where price > 5000 and protype_name = '家用电器'
```

from子句，当做一个“表”看待就可以　　where子句

性能分析 [快速编辑] [

行数： 25

+ 选项

pro_id	pro_name	protype_id	price	pinpai	chandi	protype_id	protype_name
11	索尼（SONY）60英寸全高清液晶电视	1	6999.00	索尼	北京	1	家用电器

14.4. 外连接

外连接分为左外连接和右外连接。

14.4.1. 左外连接（left join）：

语法形式：

from 表1 left [outer] join 表2 on 连接条件

说明：

1，左外连接其实是保证左边表的数据都能够取出的一种连接。

2，左外连接其实是在内连接的基础上，再加上左边表中所有不能满足条件的数据

3，关键字“outer”可以省略。

```
mysql> select * from join1 join join2 on join1.f1 = join2.c1;
+------+------+------+------+------+------+
| id   | f1   | f2   | id2  | c1   | c2   |
+------+------+------+------+------+------+
|    1 | a1   | b1   |   11 | a1   | b11  |
|    2 | a12  | b2   |   12 | a12  | b12  |
+------+------+------+------+------+------+
2 rows in set (0.00 sec)

mysql> select * from join1 left join join2 on join1.f1 = join2.c1;
+------+------+------+------+------+------+
| id   | f1   | f2   | id2  | c1   | c2   |
+------+------+------+------+------+------+
|    1 | a1   | b1   |   11 | a1   | b11  |
|    2 | a12  | b2   |   12 | a12  | b12  |
|    3 | a3   | b3   | NULL | NULL | NULL |
+------+------+------+------+------+------+
3 rows in set (0.00 sec)
```

这是内连接的结果

这是原本内连接的结果

这是无法满足内连接条件的左边表的数据结果

演示：

找出所有类别及各类别中的商品（需列出类别名称，商品名称，价格，品牌和产地）

```
38  -- 演示左连接：
39  找出所有类别及各类别中的商品（需列出类别名称，商品名称，价格，品牌和产地）
40  select protype_name, pro_name, price, pinpai, chandi
41      from product_type as t left join product as p on p.protype_id = t.protype_id
42
```

+ 选项

protype_name	pro_name	price	pinpai	chandi
家用电器	康佳（KONKA）42英寸全高清液晶电视	1999.00	康佳	深圳
家用电器	海信（Hisense）55英寸智能电视	4199.00	海信	青岛
家用电器	索尼（SONY）60英寸全高清液晶电视	6999.00	索尼	北京
手机数码	索尼（SONY）4G手机（黑色）	3238.00	索尼	深圳
手机数码	联想 双卡双待3G手机	988.00	联想	北京
电脑办公	联想（Lenovo）14.0英寸笔记本电脑	5499.00	联想	北京
电脑办公	索尼（SONY）13.3英寸触控超极本	11499.00	索尼	天津
电脑办公	联想（Lenovo）14.0英寸笔记本电脑	2999.00	联想	北京
电脑办公	惠普（HP）黑白激光打印机	1169.00	惠普	天津
图书音像	NULL	NULL	NULL	NULL
家居家具	NULL	NULL	NULL	NULL
服装配饰	NULL	NULL	NULL	NULL
个护化妆	NULL	NULL	NULL	NULL
运动户外	NULL	NULL	NULL	NULL
汽车用品	NULL	NULL	NULL	NULL
食品酒水	NULL	NULL	NULL	NULL
营养保健	NULL	NULL	NULL	NULL

14.4.2. 右外连接（right join）：

语法形式：

from 表1 right [outer] join 表2 on 连接条件

说明：

1，右外连接其实是保证右边表的数据都能够取出的一种连接。

2，右外连接其实是在内连接的基础上，再加上右边表中所有不能满足条件的数据。

3，关键字“outer”可以省略。

```
select * from join1 left join join2 on join1.f1 = join2.c1;
```

演示：

找出所有用户及其订单信息（需列出用户 id，用户名，订单号，订单总价，订单地址）

```
select user_info.user_id, user_name, order_id, order_total, order_addr
    from order_info  right join user_info on user_info.user_id = order_info.user_id;
```

user_id	user_name	order_id	order_total	order_addr
6	change	NULL	NULL	NULL
5	gynn	3	14737.00	金燕龙写字楼201室
3	shs	NULL	NULL	NULL
1	swk	1	10736.00	北京市昌平区回龙观镇金燕龙写字楼101室
1	swk	4	3999.00	金燕龙101
1	swk	5	5499.00	金燕龙写字楼前台代收
4	yhdd	NULL	NULL	NULL
2	zbj	2	5499.00	金燕龙102室

扩展一下：

```
上述语句，也可以写为如下形式，结果一样：
select user_info.user_id, user_name, order_id, order_total, order_addr
    from user_info  left join order_info on user_info.user_id = order_info.user_id;
```

可见：

左连接，右连接，其实是可以互换的——无非是把两个表的顺序调换一下。

14.5. 自连接

自连接不是一种新的连接形式，而只是一个表“自己跟自己连接”，这怎么做到呢？

语法形式：

from 表 1 as a [连接形式] join 表 1 as b on a.xx 字段 1=b.xx 字段名

说明：

1， 自连接其实还是两个表连接，只是将一个表用不同的别名，当做两个表。

2， 自连接适用于一个表中的某个字段的值“来源于”当前表的另一个字段的情况。

示例：

地区表如下所示：

id	area_name	parent_id
1	北京市	0
2	河北省	0
3	山东省	0
4	石家庄	2
5	保定	2
6	衡水	2
7	济南	3
8	青岛	3
9	烟台	3

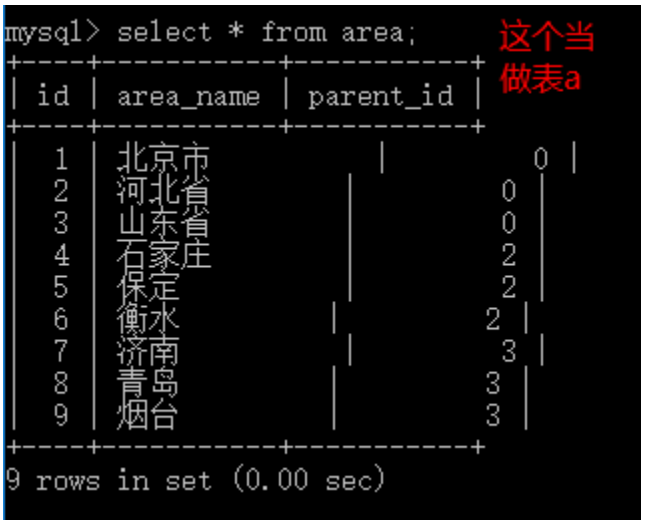
... ...

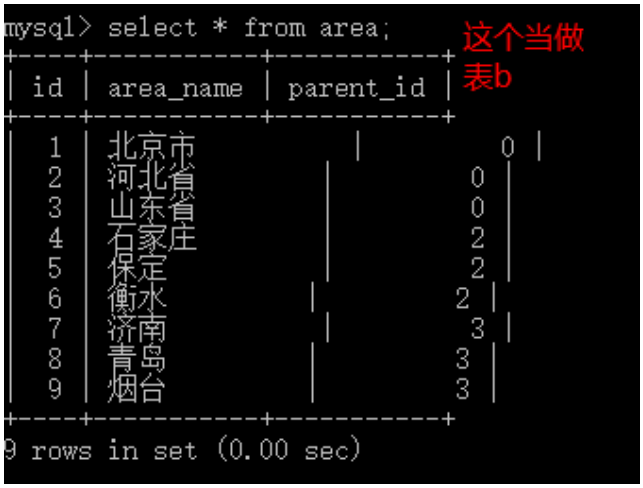
要求查询每个城市及其所在省份，结果类似如下所示：

城市　　省份

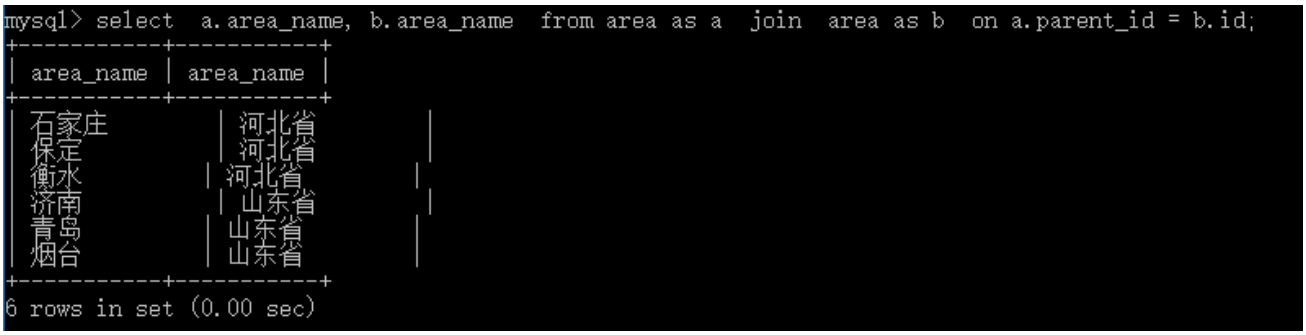
石家庄　河北省
保定　　河北省

思路：





select　a.area_name, b.area_name　from area as a　join　area as b　on a.parent_id = b.id;



需求稍作调整：

找出所有省份及其下属城市。

```
select province.area_name, city.area_name
    from area as province join area as city on province.id = city.parent_id;
-- （如果还要包括不含下属城市的省级）；
select province.area_name, city.area_name
    from area as province left join area as city on province.id = city.parent_id
    where province.parent_id = 0;
```

15. 子查询（subquery）

15.1. 子查询的概念

子查询就是指一个“正常查询语句”中的某个部分（比如 select 部分，from 部分， where 部分）又出现了查询的一种查询形式，比如：

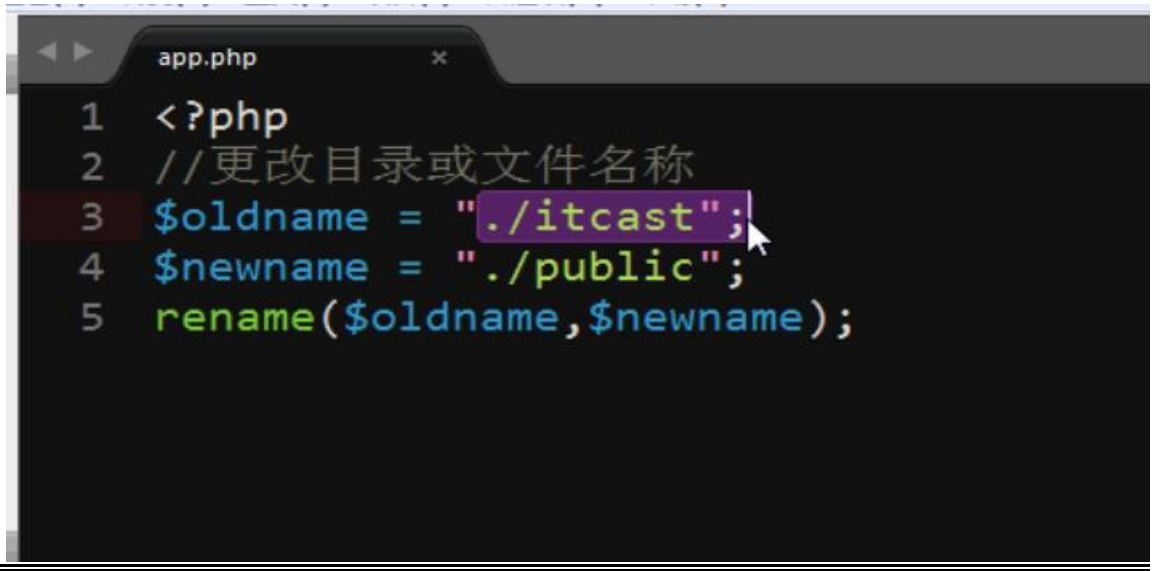
select　*　from　XX 表名　where　price >= (一个子查询语句)；

此时，子查询所在上“上层查询”，就被称为**主查询**。

也可以这么说：子查询是为主查询的某个部分提供某种数据的查询。
上一条语句中，括号中的子查询语句如果查出的是一个“某个数值”（比如 3000），则其就相当于：
select * from XX 表名 where price >=3000；

15.2. 标量子查询

- 含义：
  标量子查询就是指子查询的结果是“单个值”（一行一列）的查询。
- 使用：
  标量子查询通常用在 where 子句中，作为主查询的一个条件判断的数据。
  本质上，标量子查询的结果，就可以直接当做“一个值”来使用。



示例：

找出产品表中价格大于北京产地的产品平均价的所有产品。

```
67  select * from product where price > (
68          #查询出北京产地所有产品的平均价
69      select avg(price) from product where chandi='北京'
70      );
```

```
mysql> select * from product where price > (
    -> #查询出北京产地所有产品的平均价
    -> select avg(price) from product where chandi='北京'
    -> );
+--------+-----------------------------------+-----------+---------+--------+--------+
| pro_id | pro_name                          | protype_id | price   | pinpai | chandi |
+--------+-----------------------------------+-----------+---------+--------+--------+
|      3 | 海信（Hisense）55英寸智能电视     |         1 | 4199.00 | 海信   | 青岛   |
|      4 | 联想（Lenovo）14.0英寸笔记本电脑  |         3 | 5499.00 | 联想   | 北京   |
|      5 | 索尼（SONY）13.3英寸触控超极本    |         3 | 11499.00 | 索尼  | 天津   |
|     11 | 索尼（SONY）60英寸全高清液晶电视  |         1 | 6999.00 | 索尼   | 北京   |
+--------+-----------------------------------+-----------+---------+--------+--------+
4 rows in set (0.00 sec)
```

课堂练习：
找出所有奢侈品！

奢侈品的定义为：其价格超过贵重品的平均价！

贵重品的定义为：超过所有商品的平均价！

```
76  select * from product where price > (
77      #找出贵重品的平均价
78      select avg(price) from product where price > (
79          #去找出所有商品的平均价
80          select avg(price) from product
81      )
82  );
```

```
+--------+------------------------------+------------+----------+--------+--------+
| pro_id | pro_name                     | protype_id | price    | pinpai | chandi |
+--------+------------------------------+------------+----------+--------+--------+
|      5 | 索尼（SONY）13.3英寸触控超极本 |          3 | 11499.00 | 索尼   | 天津
+--------+------------------------------+------------+----------+--------+--------+
1 row in set (0.00 sec)
```

15.3. 列子查询

- 含义：

  列子查询查出的结果为“一列数据”，类似这样：

  select pinpai from product where chandi = ‘北京’;

  结果为：

  ```
  +--------+
  | pinpai |
  +--------+
  | 联想   |
  | 索尼   |
  | 联想   |
  | 联想   |
  +--------+
  ```

- 使用：

  列子查询通常用在 where 子句的 in 运算符中，代替 in 运算符中的“字面值”列表数据。

  比如：

  select * from product where chandi in ('北京', '深圳','天津')

  如果 in 中的数据并不方便一个一个列出，但可以通过一个查询得到，就可以使用查询来实现：

  select * from product where chandi in (select chandi from product price > 4000);

示例：

查出出产贵重商品（假设价格超过 5000 即为贵重商品）的那些产地的所有商品。

```
mysql> select * from product where chandi in(
    -> #找出出产贵重品的那些产地
    -> select chandi from product where price > 5000
    -> );
+--------+--------------------------------+------------+---------+--------+--------+
| pro_id | pro_name                       | protype_id | price   | pinpai | chandi |
+--------+--------------------------------+------------+---------+--------+--------+
|      4 | 联想（Lenovo）14.0英寸笔记本电脑 |          3 | 5499.00 | 联想   | 北京   |
|      5 | 索尼（SONY）13.3英寸触控超极本   |          3 | 11499.00 | 索尼  | 天津   |
|     11 | 索尼（SONY）60英寸全高清液晶电视 |          1 | 6999.00 | 索尼   | 北京   |
|     12 | 联想（Lenovo）14.0英寸笔记本电脑 |          3 | 2999.00 | 联想   | 北京   |
|     13 | 联想 双卡双待3G手机              |          2 | 988.00  | 联想   | 北京   |
|     15 | 惠普（HP）黑白激光打印机         |          3 | 1169.00 | 惠普   | 天津   |
+--------+--------------------------------+------------+---------+--------+--------+
6 rows in set (0.00 sec)
```

15.4. 行子查询

- 含义：
  行子查询查出的结果通常是一行，类似这样：
  select pinpai, chandi from product where price=11499;
  结果为：

```
+--------+--------+
| pinpai | chandi |
+--------+--------+
| 联想   | 北京   |
+--------+--------+
```

- 使用：
  行子查询的结果通常跟“**行构造符**”一起，在 where 条件子句中做为条件数据，类似这样：
  where （字段 1, 字段 2） = （行子查询）
  或
  where row(字段 1, 字段 2） = （行子查询） //含义跟上一行是一样的，即 row 可以省略

示例：

找出跟单价最高的商品同品牌同产地的所有商品。

```
另一个分析思路：
1，先找出单价最高的商品的品牌和产地
    select pinpai, chandi from product where price = (
            select max(price) from product
        )
2，然后，在上述“已知”的产地和品牌的情况，找同产地和品牌的“所有商品”
select * from product where row(pinpai, chandi) = (
    select pinpai, chandi from product where price = (
        select max(price) from product
    )
);
```

```
+--------+-----------------------------------+-----------+----------+--------+--------+
| pro_id | pro_name                          | protype_id | price    | pinpai | chandi |
+--------+-----------------------------------+-----------+----------+--------+--------+
|      5 | 索尼（SONY）13.3英寸触控超极本                |         3 | 11499.00 | 索尼     | 天津
|     11 | 索尼（SONY）60英寸全高清液晶电视                |         1 |  6999.00 | 索尼     | 天津
+--------+-----------------------------------+-----------+----------+--------+--------+
2 rows in set (0.00 sec)
```

15.5. 表子查询

- 含义：
  当一个子查询查出的结果是“多行多列”的时候，就是表子查询。
  表子查询的结果相当于一个表，可以直接当做一个表来使用。
- 使用：
  表子查询通常用在主查询的 from 子句中，作为一个“数据源”。
  注意：
  此时需要给该子查询设置一个别名，类似这样：
  from (select ... 子查询) as tab1

示例：

查出商品价格大于 4000 的所有商品的数量和均价

```
#1先找出价格大于4000的所有商品：
select * from product where price > 4000;
#2将上述查询结果（是多行多列的）当做一个表（表子查询），
#去对这些数据求出总数量和均价
select count(*) as 总数, avg(price) as 平均价 from (
    select * from product where price > 4000
) as t;
当然，本需求，不用子查询，也可以实现，如下：
select count(*) as 总数, avg(price) as 平均价 from product where price>4000;
```

两个结果一样:

```
+------+-------------+
| 总数     | 平均价
+------+-------------+
|    4 | 7049.000000 |
+------+-------------+
1 row in set (0.00 sec)
```

15.6. 有关子查询的特定关键字

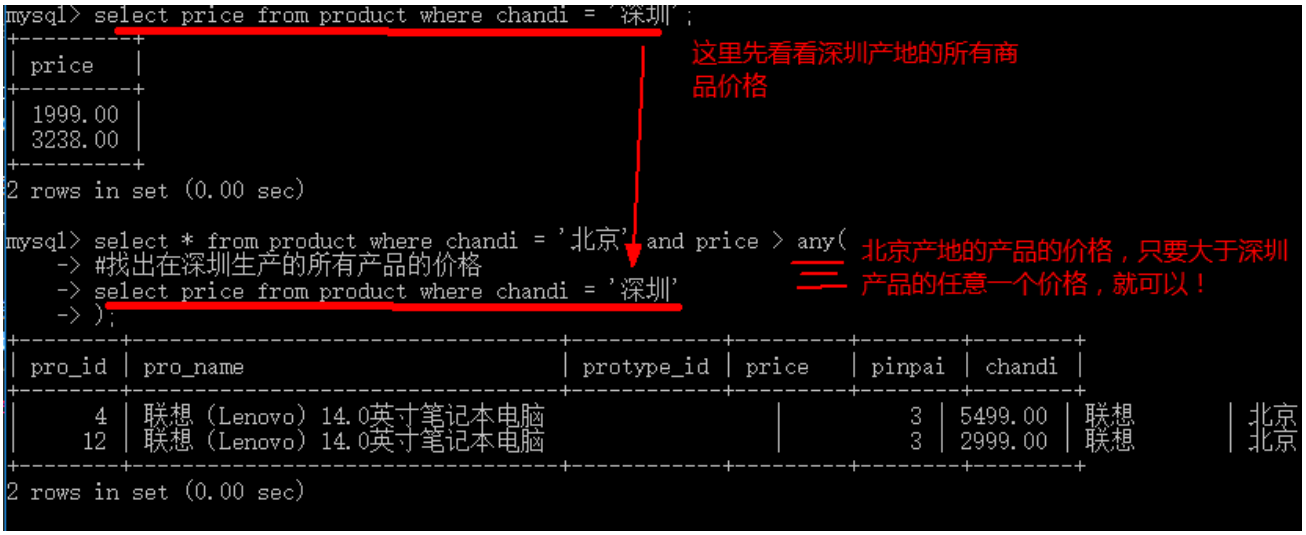
15.6.1. in 关键字

in 关键字在子查询中主要用在列子查询中代替人为手工罗列出来的多个“字面值”数据。
举例：
找出联想品牌的商品都有哪些类别。
select * from product_type where protype_id in(

#找出联想品牌的所有商品的类别 id
select distinct protype_id from product where pinpai = '联想'
)

15.6.2. any 关键字

any 关键字用在比较操作操符的后面，表示查询结果的多个数据中的任一个满足该比较操作符就算满足。
举例：
找出在北京生产的但价格比在深圳生产的任一商品贵的商品。




```
mysql> select price from product where chandi = '深圳';
+---------+
| price   |
+---------+
| 1999.00 |
| 3238.00 |
+---------+
2 rows in set (0.00 sec)

mysql> select * from product where chandi = '北京' and price > any(
    -> #找出在深圳生产的所有产品的价格
    -> select price from product where chandi = '深圳'
    -> );
+--------+-----------------------------------+------------+---------+--------+---------+
| pro_id | pro_name                          | protype_id | price   | pinpai | chandi  |
+--------+-----------------------------------+------------+---------+--------+---------+
|      4 | 联想 (Lenovo) 14.0英寸笔记本电脑  |          3 | 5499.00 | 联想   | 北京    |
|     12 | 联想 (Lenovo) 14.0英寸笔记本电脑  |          3 | 2999.00 | 联想   | 北京    |
+--------+-----------------------------------+------------+---------+--------+---------+
2 rows in set (0.00 sec)
```

15.6.3. all 关键字

all 关键字用在比较操作操符的后面，表示查询结果的多个数据中的所有都满足该比较操作符才算满足。
举例：
找出在北京生产的但价格比在深圳生产的所有商品都贵的商品。

```
-- all 的使用：
示例：找出在北京生产的但价格比在深圳生产的所有商品都贵的商品。
select * from product where chandi = '北京' and price > all(
    #找出在深圳生产的所有产品的价格
    select price from product where chandi = '深圳'
);
```

```
+--------+-----------------------------------+------------+---------+--------+---------+
| pro_id | pro_name                          | protype_id | price   | pinpai | chandi  |
+--------+-----------------------------------+------------+---------+--------+---------+
|      4 | 联想 (Lenovo) 14.0英寸笔记本电脑  |          3 | 5499.00 | 联想   | 北京    |
+--------+-----------------------------------+------------+---------+--------+---------+
1 row in set (0.00 sec)
```

15.7. exists 子查询

- 形式：
  where exists (任何子查询)

- 含义：
  该子查询如果“有数据”， 则该 exists()的结果为“true”， 即相当于 where true （恒真）
  该子查询如果“没有数据”，则该 exists()的结果为“false”，即相当于 where false（恒假）

- 说明：
  1，此子查询语句通常需要用到主查询语句中的字段作为查询条件。

- 示例：
  1，查询商品分类名称中带“电”字的所有商品；
  2，查询联想品牌的产品都有哪些分类；

```
1，查询商品分类名称中带“电”字的所有商品；
select * from product where exists(
    select * from product_type where protype_name like '%电%' and product_type.protype_id = product.protype_id
);

2，查询联想品牌的产品都有哪些分类；
select * from product_type where exists(
    select * from product where pinpai = '联想' and product.protype_id = product_type.protype_id
);
```

特别注意：
通常，有意义 exists 子查询不能单独执行
对比：之前的 4 种子查询都可以单独执行

16. 数据管理

16.1. 数据备份

数据备份就是指将一个数据库中的数据，转存为一个或多个文件的过程。

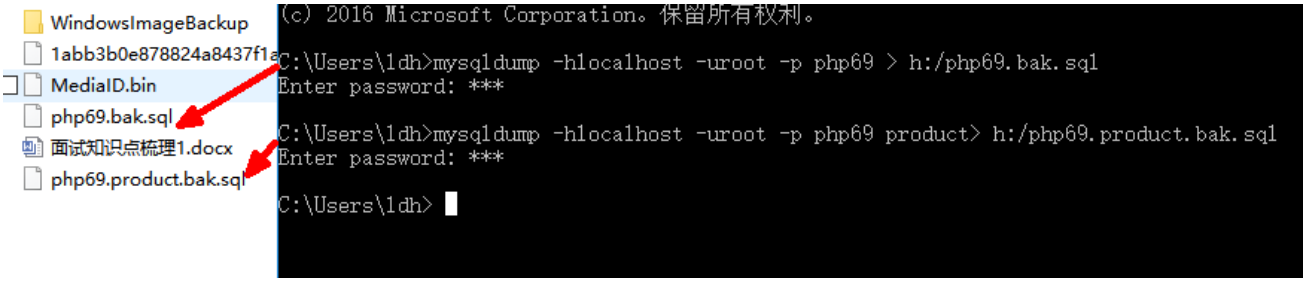
16.1.1. 备份整个数据库

- 命令形式：
  mysqldump.exe -h 主机地址 -u 用户名 -p 密码 数据库名 > 备份文件名(含路径)

- 说明：
  1， 跟登录 mysql 类似，密码可以不写，则随后会提示输入
  2， 该语句是 mysql/bin 中的一个命令，不是 sql 语句（即不应该登录 mysql 后使用）

- 示例：

备份我们的 php69 数据库：

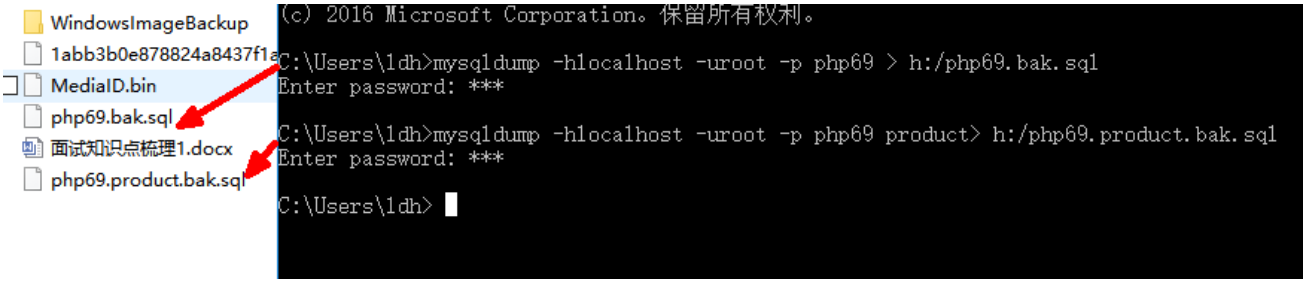


16.1.2. 备份单个表

- 命令形式：
  mysqldump.exe -h 主机地址 -u 用户名 -p 密码 数据库名 表名 > 备份文件名(含路径)

- 说明：
  1， 跟登录 mysql 类似，密码可以不写，则随后会提示输入
  2， 该语句是 mysql/bin 中的一个命令，不是 sql 语句（即不应该登录 mysql 后再去使用）

- 示例：

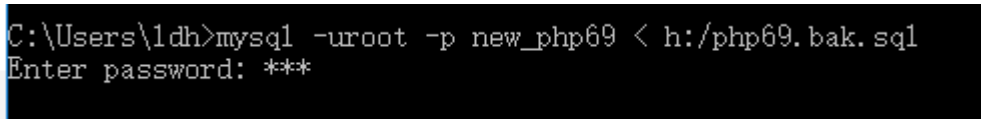


16.2. 数据还原（数据恢复）

数据还原（恢复）是指将一个之前备份过的数据文件，恢复（还原）到某个数据库的过程。
还原其实不分整个库还是单个表，都是一样的。

- 命令形式：
  mysql.exe -h 主机地址 -u 用户名 -p 密码 目标数据库名 < 想要还原的备份文件名(含路径)

- 示例：



17. 用户管理：

用户管理主要包括两方面的工作：

用户账号的管理，包括：创建，删除，改密

用户权限的管理，包括：授予权限，取消权限

17.1. 查看用户

mysql 数据库管理系统中有个数据库叫做“mysql”，绝对不能删除！

其中有个表“user”，就是存储了当前数据库系统中的所有用户信息。

初始时只有一个用户：root 。

查看用户：

```
use mysql;
select * from user;
```

17.2. 创建用户

语法形式：

```
create user ‘用户名’[@’允许登录的地址’] identified by ‘密码’;
```

说明：

1， 创建用户之后，数据库 mysql 中的 user 表中就会多一个用户。

2，‘允许登录的地址’，就是允许登录的客户端的 ip 地址，或

① ”localhost”表示只能本地登录；

② “%”表示任何位置都可以登录；

③ 该部分可以省略，如果省略，默认就是”%”；

④ 后续涉及到用户的操作，都是这个格式。

示例：

```
create user ‘user1’ identified by ‘123’;
create user ‘user2’@’localhost’ identified by ‘123’;
create user ‘user3’@’192.168.1.103’ identified by ‘123’;
create user ‘user4’@’%’ identified by ‘123’;
```

```
mysql> create user 'user_for_all' identified by '123';
Query OK, 0 rows affected (0.00 sec)

mysql> create user 'onlyforyou'@'192.168.149.40' identified by '123';
Query OK, 0 rows affected (0.00 sec)
```

17.3. 删除用户

语法形式：

drop user 用户[@'允许登录的地址'];

删除用户后，数据库 mysql 中的 user 表中就会少一个用户。

```
mysql> drop user onlyforyou@192.168.149.40;
Query OK, 0 rows affected (0.00 sec)
```

17.4. 修改/设置用户密码

语法形式：

set password for 用户[@'允许登录的地址'] = password('密码');

17.5. 授予用户权限

语法形式：

grant 操作 1，操作 2，.... on *.*或数据库名.* 或 数据库名.表名 to 用户[@'允许登录的地址'];

说明：

1，“操作”其实就是权限名，是一个“特定词”，比如：delete, insert, update, select, create, 等等。

① 其中，还可以用“all”，表示“所有权限”（除了 grant 权限）。

2，on 后表示对“什么东西”来设定该权限，意思是对什么库的什么表，其中：

① *.* ：表示所有库的所有表；

② 数据库名.* ：表示该指定数据库的所有表；

③ 数据库名.表名：表示该指定数据库的该指定表；

示例：

```
mysql> grant select on php69.* to user_for_all;
Query OK, 0 rows affected (0.00 sec)
```

- **mysql 中的所有操作（权限），有如下所示：**

17.6. 取消用户授权

语法形式：

revoke 操作 1，操作 2，.... on *.*或数据库名.* 或 数据库名.表名 from 用户[@'允许的地址'];

```
mysql> revoke select on php69.* from user_for_all;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

提示：

测试的时候，需要使用两个 cmd 窗口，一个是 root 用户进行用户和权限管理，另一个窗口用于测试。