

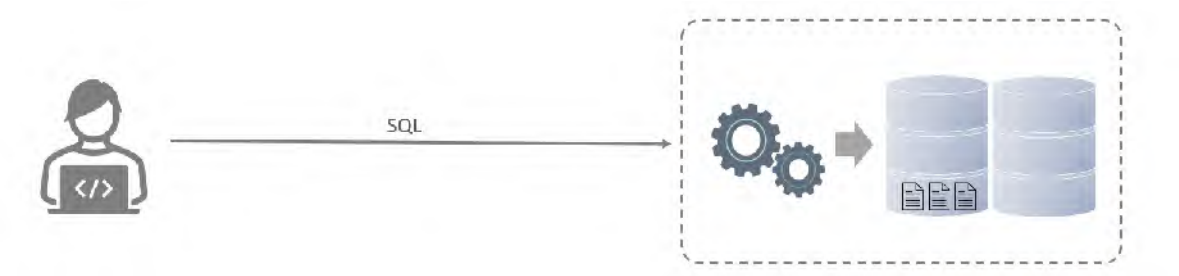
1. MySQL概述

在这一章节，我们主要介绍两个部分，数据库相关概念及MySQL数据库的介绍、下载、安装、启动及连接。

1.1 数据库相关概念

在这一部分，我们先来讲解三个概念：数据库、数据库管理系统、SQL。

名称	全称	简称
数据库	存储数据的仓库，数据是有组织的进行存储	DataBase (DB)
数据库管理系统	操纵和管理数据库的大型软件	DataBase Management System (DBMS)
SQL	操作关系型数据库的编程语言，定义了一套操作关系型数据库统一标准	Structured Query Language (SQL)



而目前主流的关系型数据库管理系统的市场占有率排名如下：

Rank			DBMS	Database Model	Score		
Jan 2022	Dec 2021	Jan 2021			Jan 2022	Dec 2021	Jan 2021
1.	1.	1.	Oracle +	Relational, Multi-model ⓘ	1266.89	-14.85	-56.05
2.	2.	2.	MySQL +	Relational, Multi-model ⓘ	1206.05	+0.01	-46.01
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model ⓘ	944.81	-9.21	-86.42
4.	4.	4.	PostgreSQL + ⓘ	Relational, Multi-model ⓘ	606.56	-1.66	+54.33
5.	5.	5.	IBM Db2	Relational, Multi-model ⓘ	164.20	-2.98	+7.03
6.	↑ 7.	↑ 7.	Microsoft Access	Relational	128.95	+2.96	+13.61
7.	↓ 6.	↓ 6.	SQLite +	Relational	127.43	-1.25	+5.54
8.	8.	8.	MariaDB +	Relational, Multi-model ⓘ	106.42	+2.06	+12.63
9.	9.	↑ 10.	Microsoft Azure SQL Database	Relational, Multi-model ⓘ	86.32	+3.07	+14.96
10.	10.	↑ 11.	Hive +	Relational	83.45	+1.52	+13.02

- Oracle：大型的收费数据库，Oracle公司产品，价格昂贵。
- MySQL：开源免费的中小型数据库，后来Sun公司收购了MySQL，而Oracle又收购了Sun公司。
目前Oracle推出了收费版本的MySQL，也提供了免费的社区版本。

- SQL Server: Microsoft 公司推出的收费的中型数据库, C#、.net等语言常用。
- PostgreSQL: 开源免费的中小型数据库。
- DB2: IBM公司的大型收费数据库产品。
- SQLite: 嵌入式的微型数据库。Android内置的数据库采用的就是该数据库。
- MariaDB: 开源免费的中小型数据库。是MySQL数据库的另外一个分支、另外一个衍生产品, 与MySQL数据库有很好的兼容性。

而不论我们使用的是上面的哪一个关系型数据库, 最终在操作时, 都是使用SQL语言来进行统一操作, 因为我们前面讲到SQL语言, 是操作关系型数据库的 **统一标准**。所以即使我们现在学习的是MySQL, 假如我们以后到了公司, 使用的是别的关系型数据库, 如: Oracle、DB2、SQLServer, 也完全不用担心, 因为操作的方式都是一致的。

1.2 MySQL数据库

1.2.1 版本



官方: <https://www.mysql.com/>

MySQL官方提供了两种不同的版本:

- 社区版本 (MySQL Community Server)
免费, MySQL不提供任何技术支持
- 商业版本 (MySQL Enterprise Edition)
收费, 可以使用30天, 官方提供技术支持

本课程采用的是MySQL最新的社区版-MySQL Community Server 8.0.26

1.2.2 下载


下载地址: <https://downloads.mysql.com/archives/installer/>

MySQL Product Archives

MySQL Installer (Archived Versions)

Product Version: **8.0.26** 
Operating System: **Microsoft Windows** 

Windows (x86, 32-bit), MSI Installer <small>[mysql-installer-community-8.0.26.0.msi]</small>	Jul 6, 2021	2.4M	Download <small>MD5: 6244295b14e5273e1b5b1249a1460fb Signature</small>
Windows (x86, 32-bit), MSI Installer <small>[mysql-installer-community-8.0.26.0.msi]</small>	Jul 6, 2021	450.7M	Download <small>MD5: 1b93e0c73121143e81724e019c816 Signature</small>

 We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

MySQL open source software is provided under the GPL License.


也可以使用课程资料中提供的MySQL的安装包:

 **mysql-installer-community-8.0.26.0.msi** Windows Installer 程序包 461,472 KB

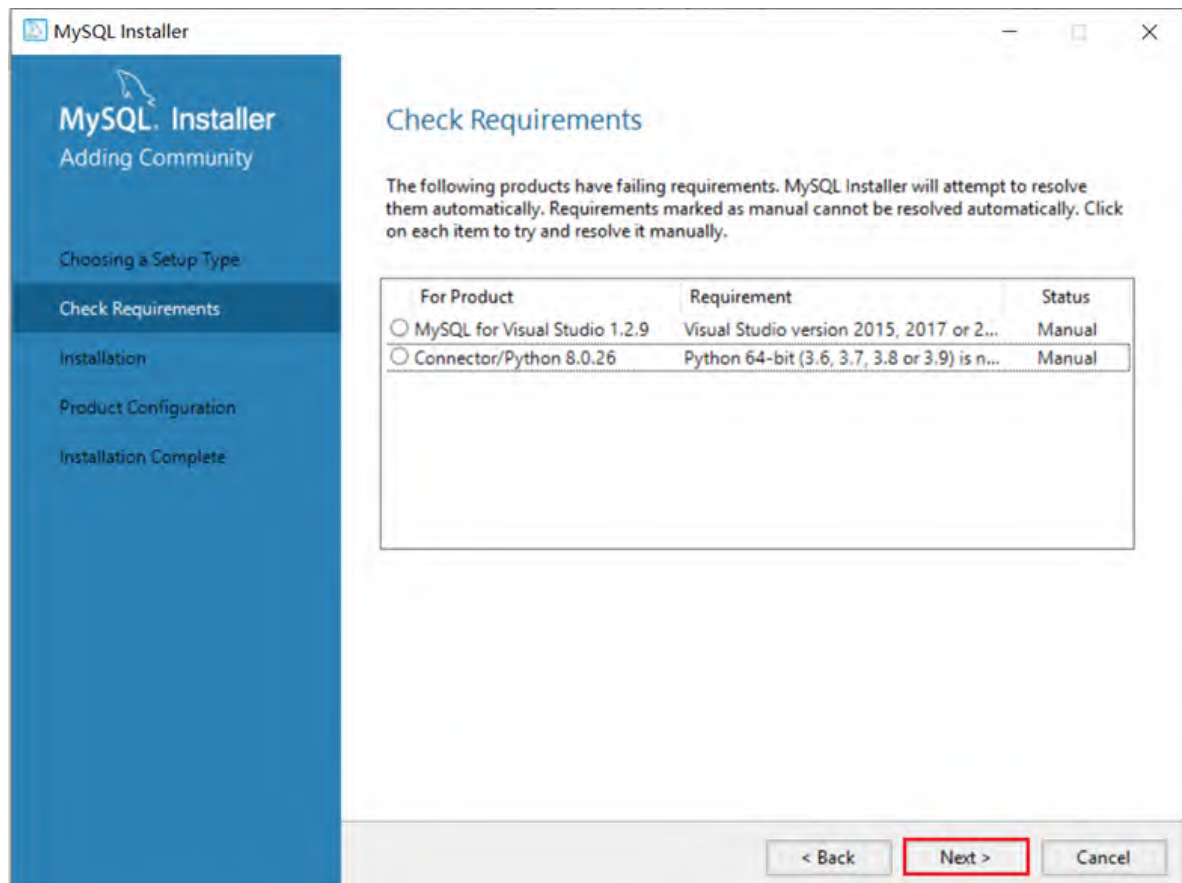
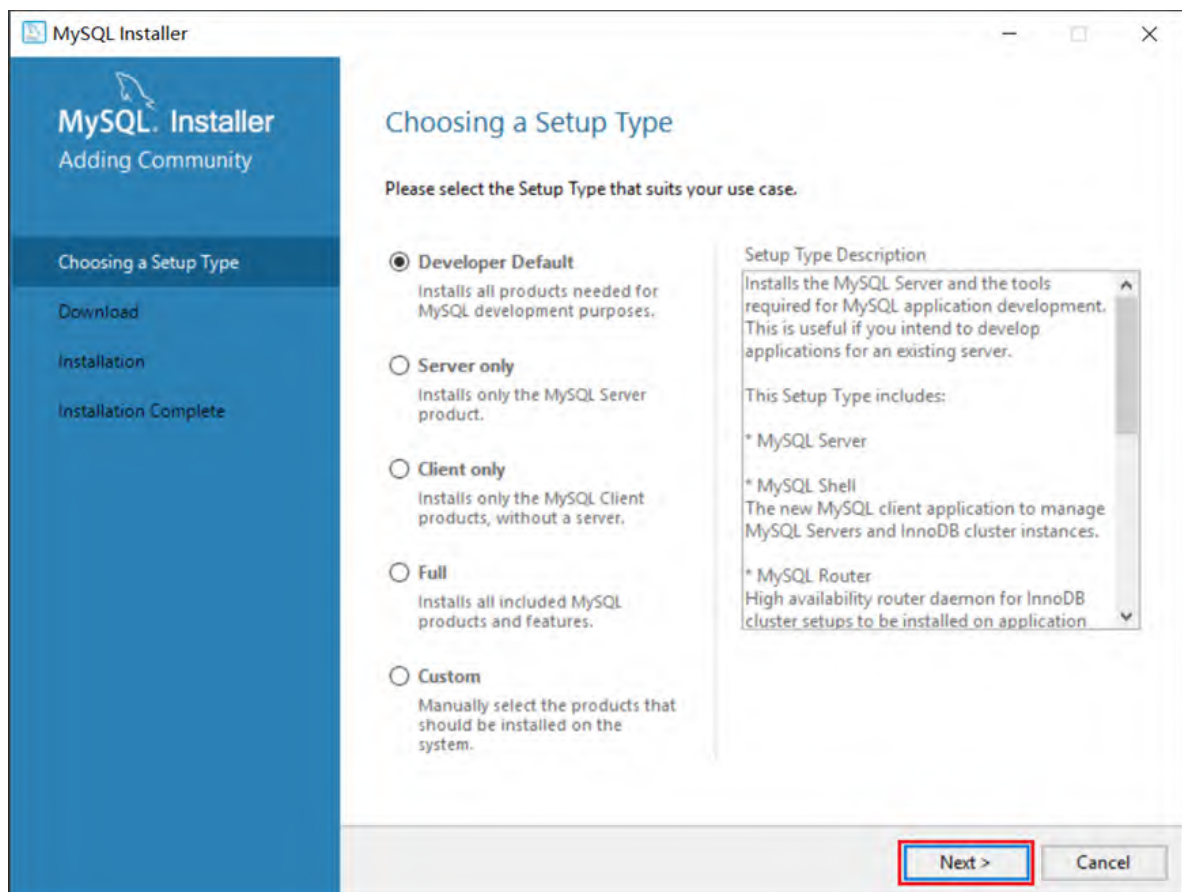
1.2.3 安装

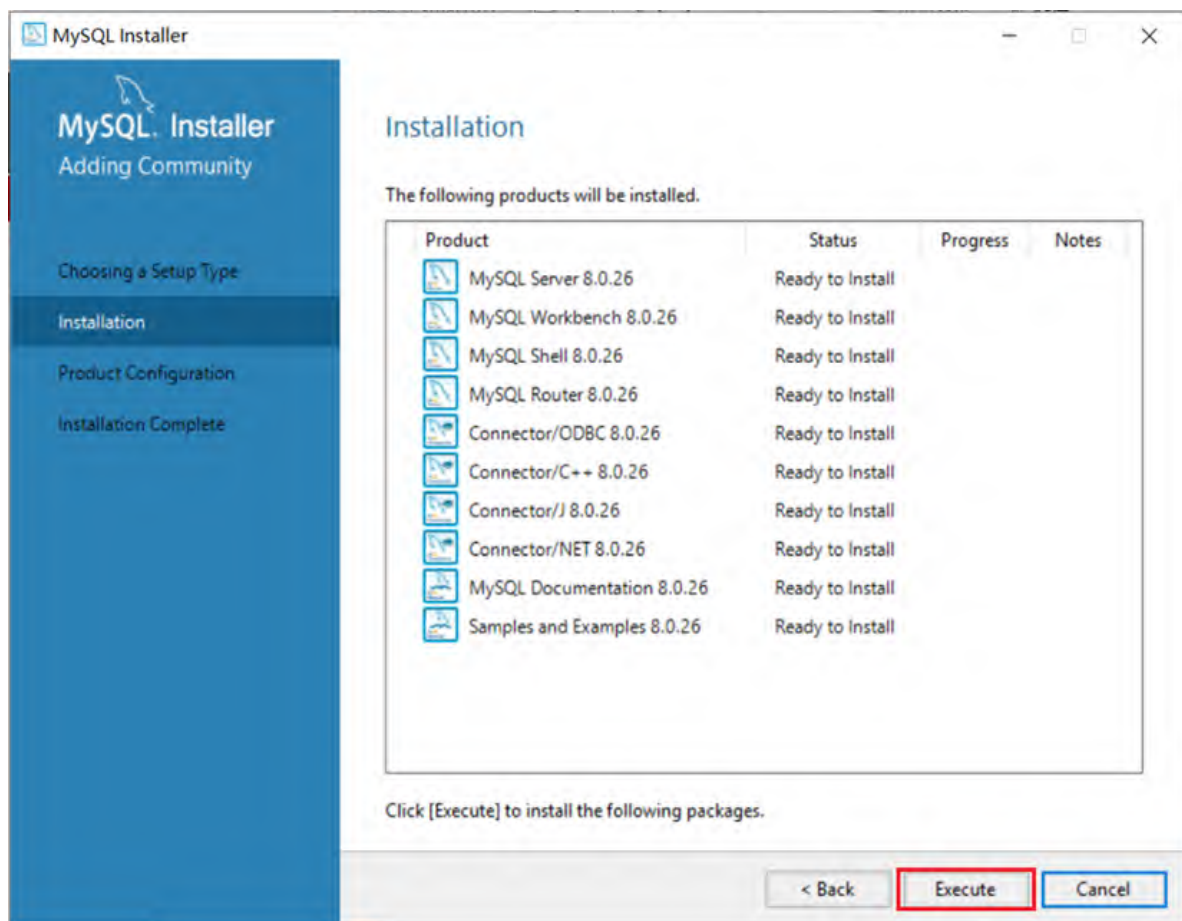
要想使用MySQL, 我们首先先得将MySQL安装好, 我们可以根据下面的步骤, 一步一步的完成MySQL的安装。

1). 双击官方下来的安装包文件

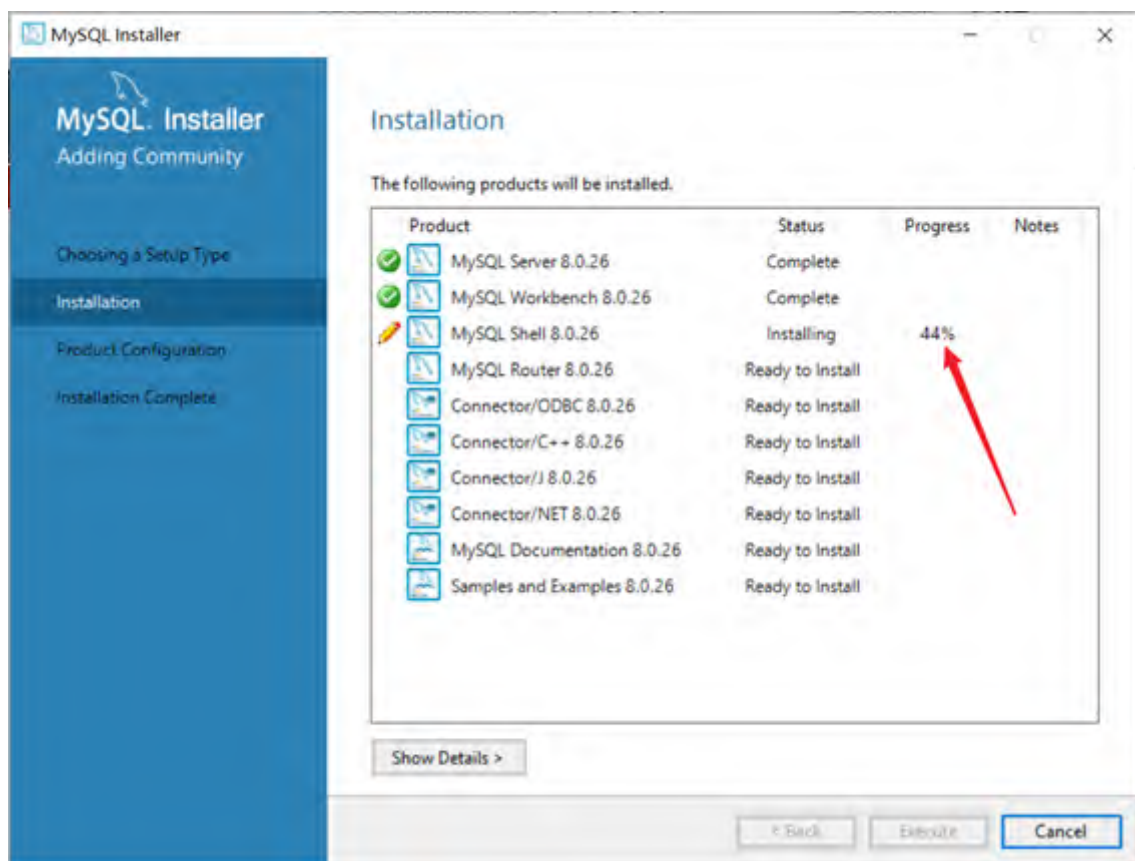
 **mysql-installer-community-8.0.26.0.msi**

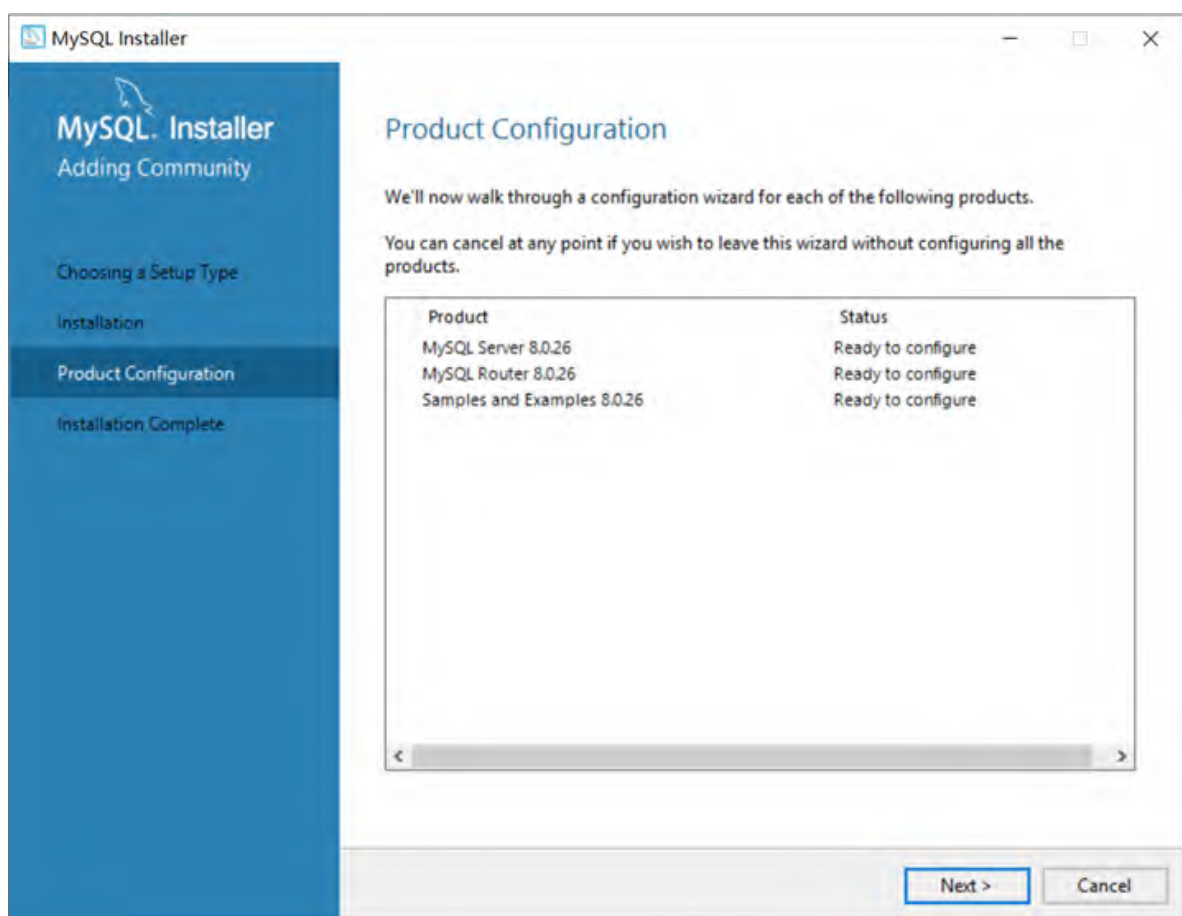
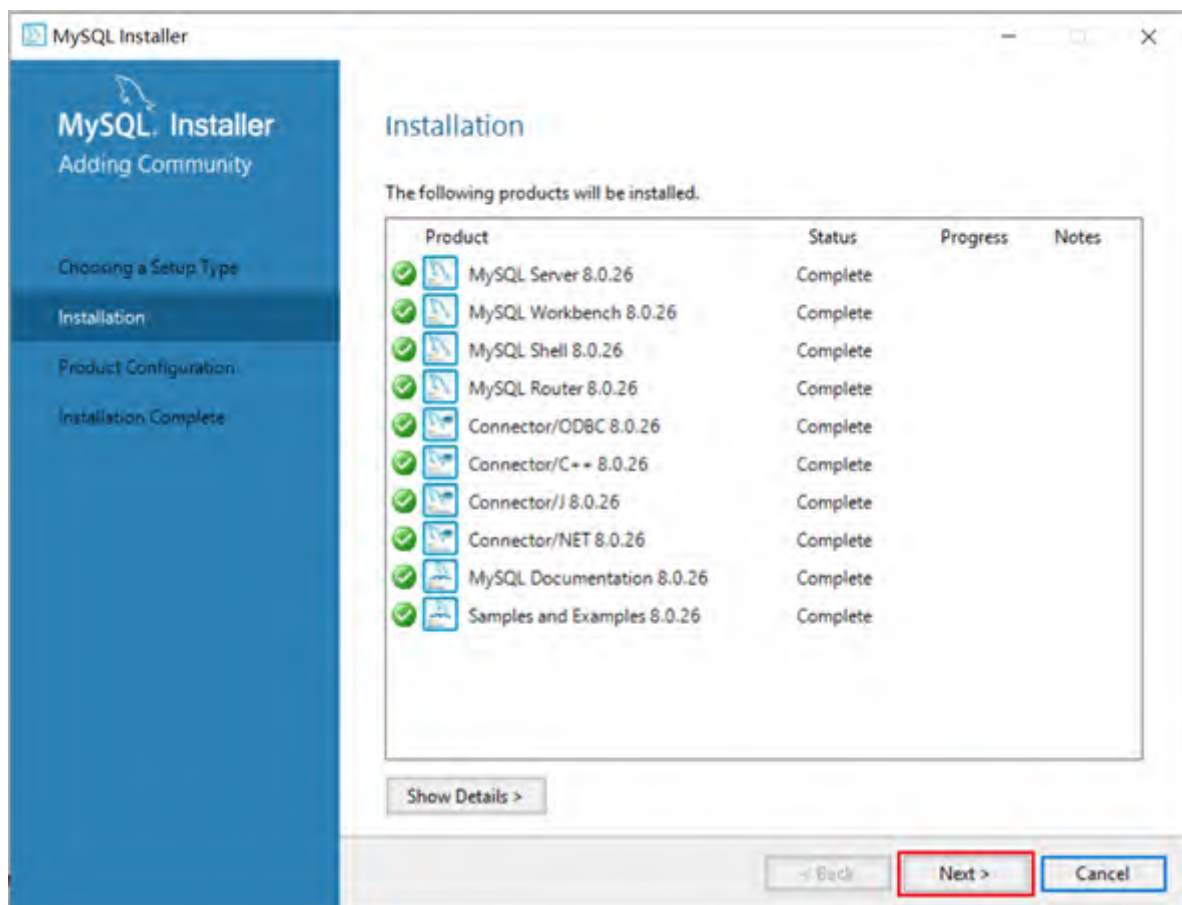
2). 根据安装提示进行安装

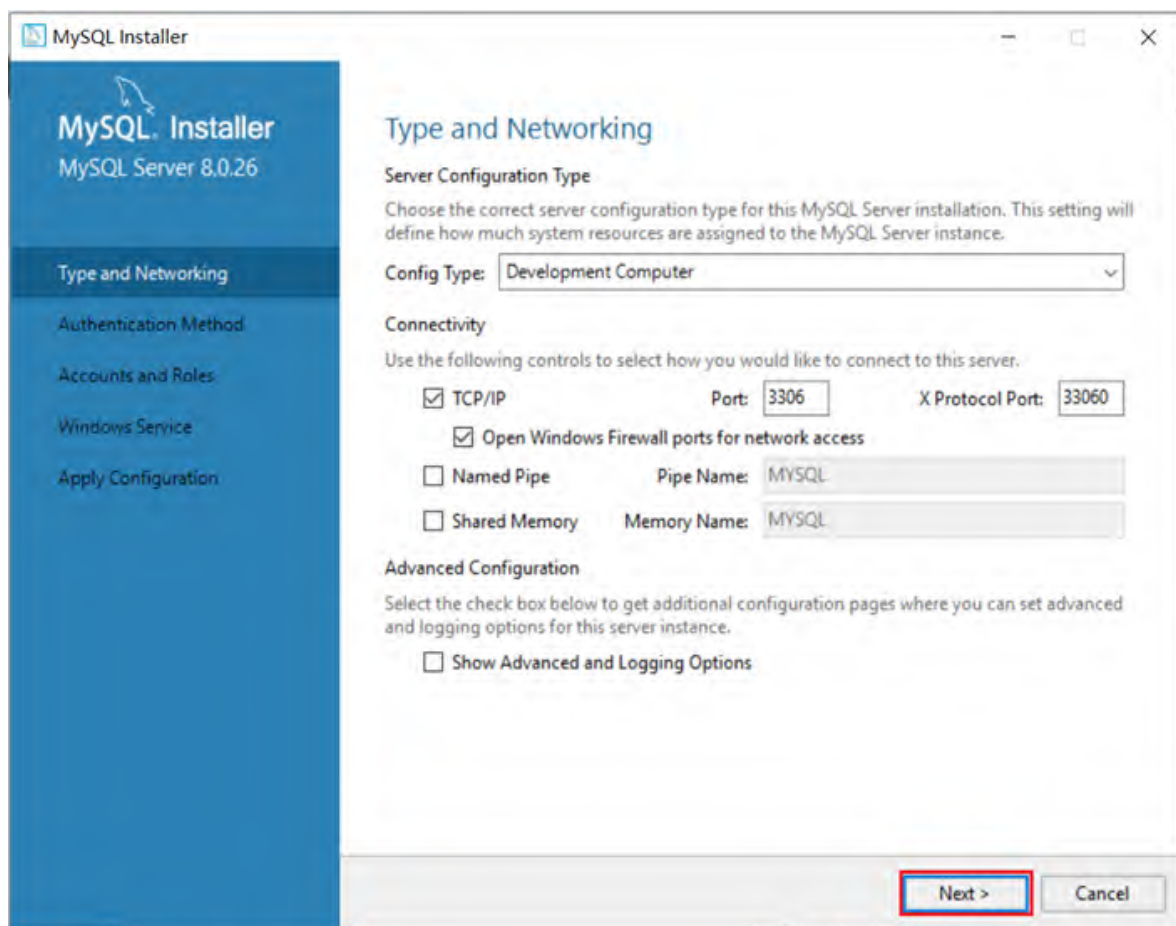




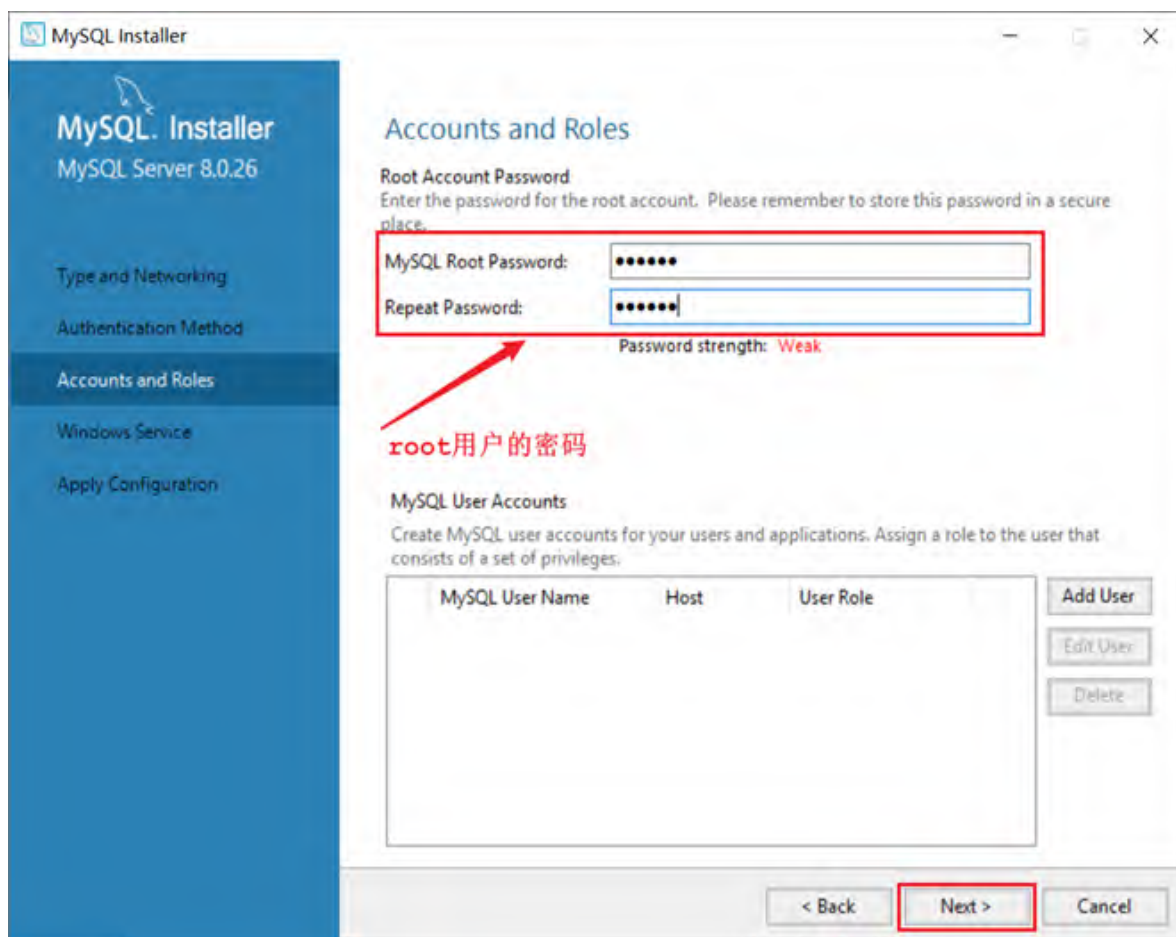
安装MySQL的相关组件，这个过程可能需要耗时几分钟，耐心等待。

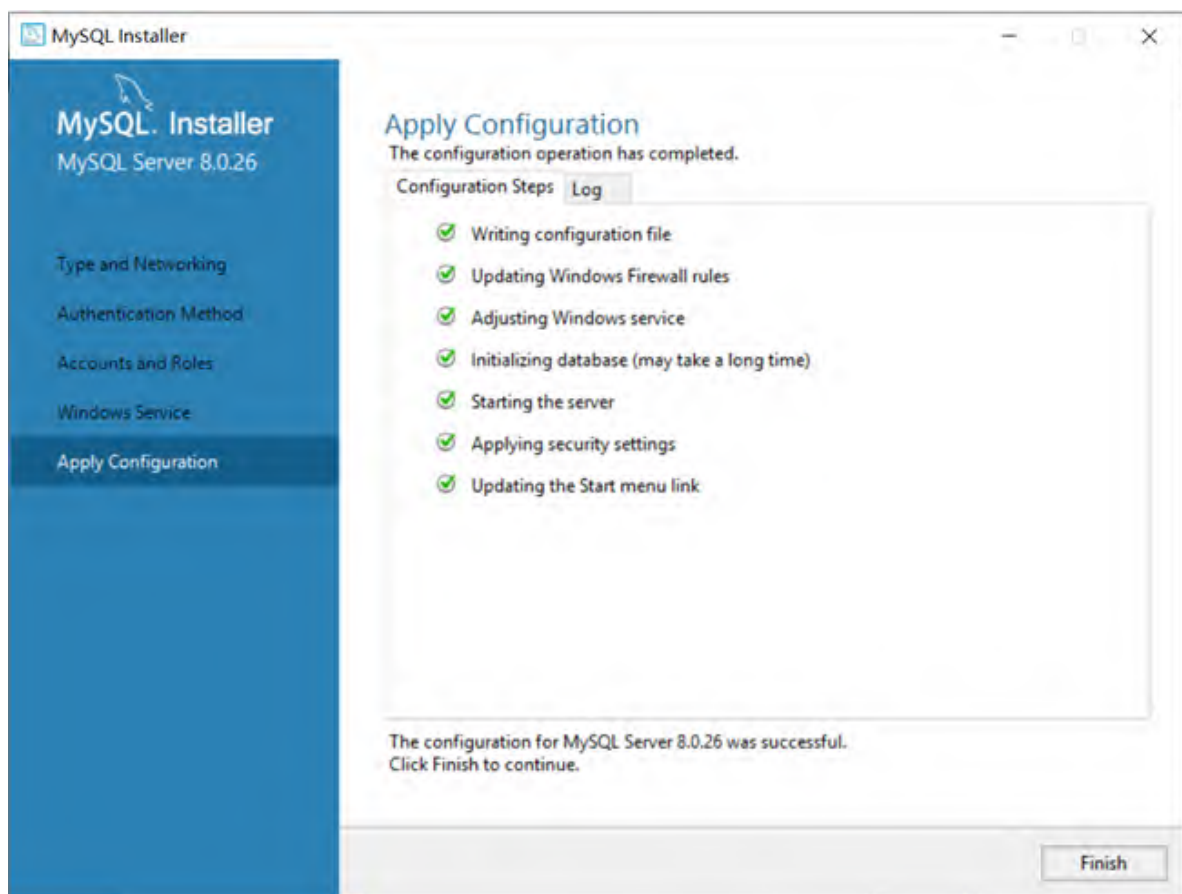
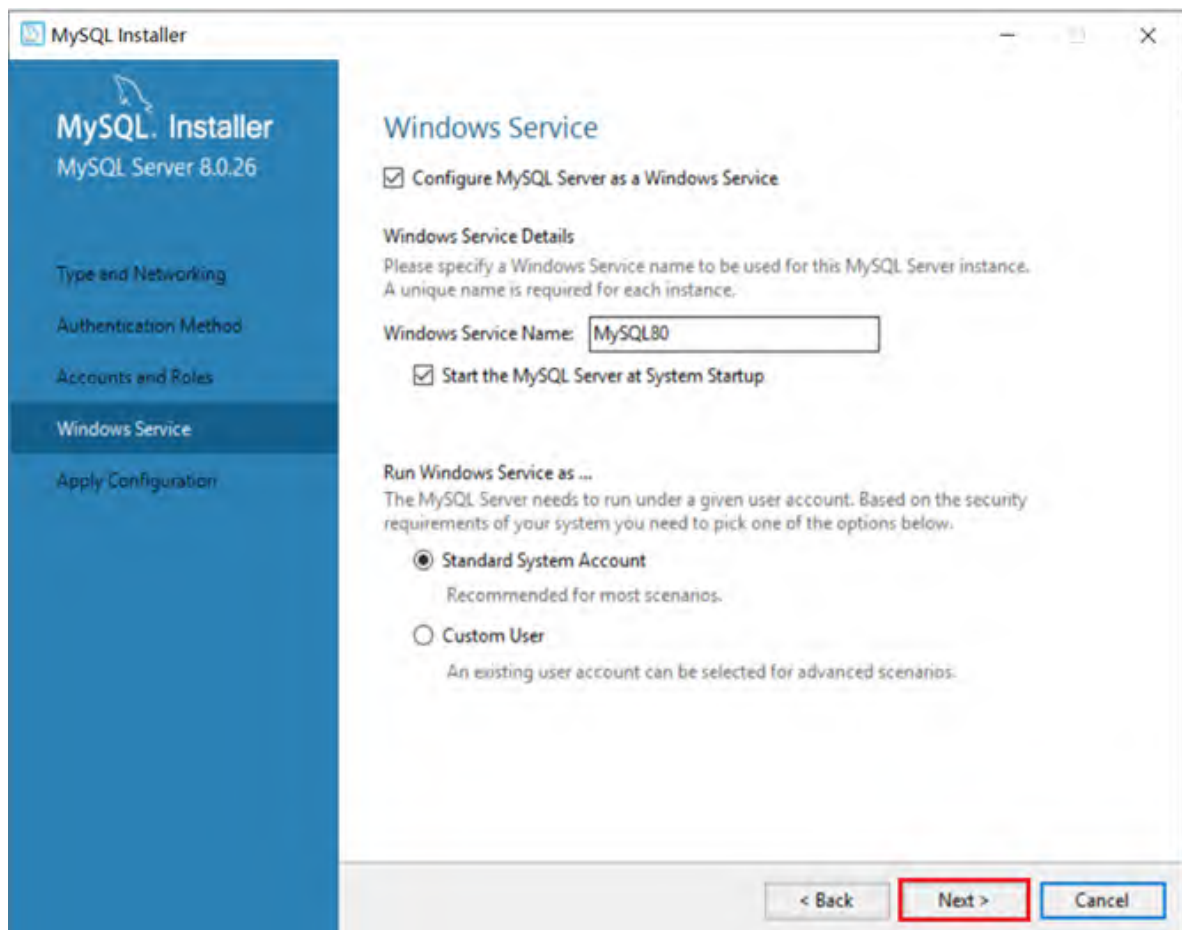






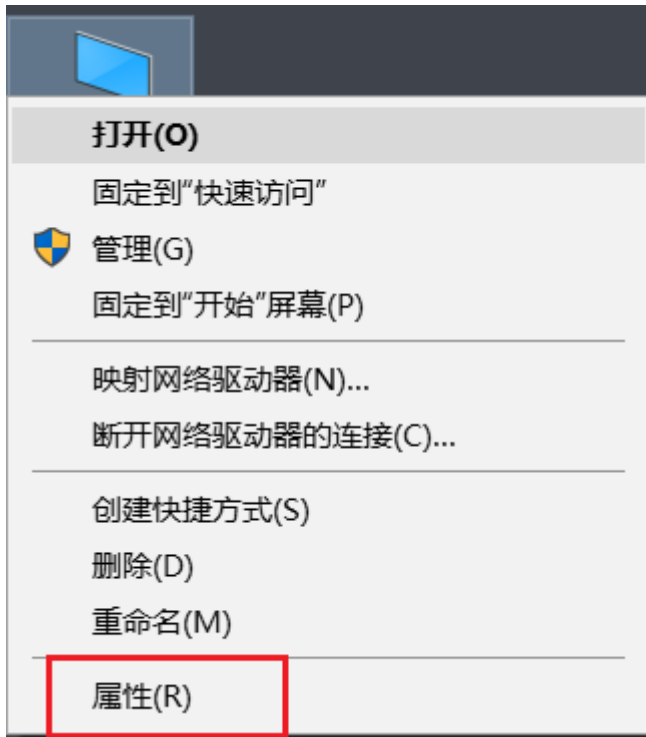
输入MySQL中root用户的密码，一定记得记住该密码



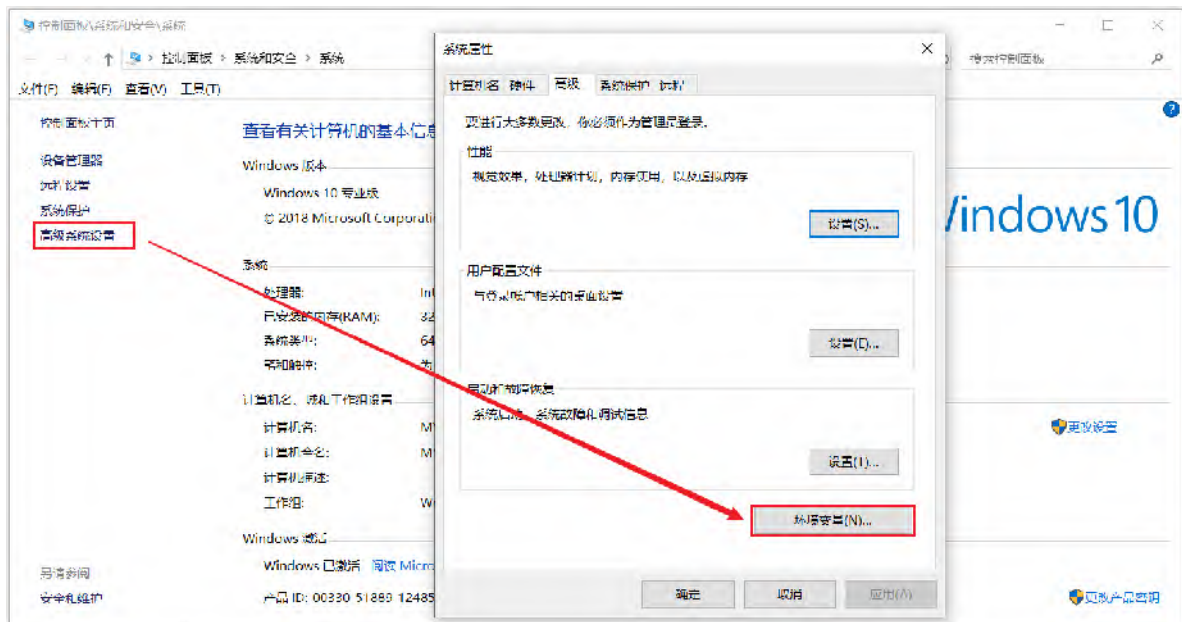


安装好MySQL之后，还需要配置环境变量，这样才可以在任何目录下连接MySQL。

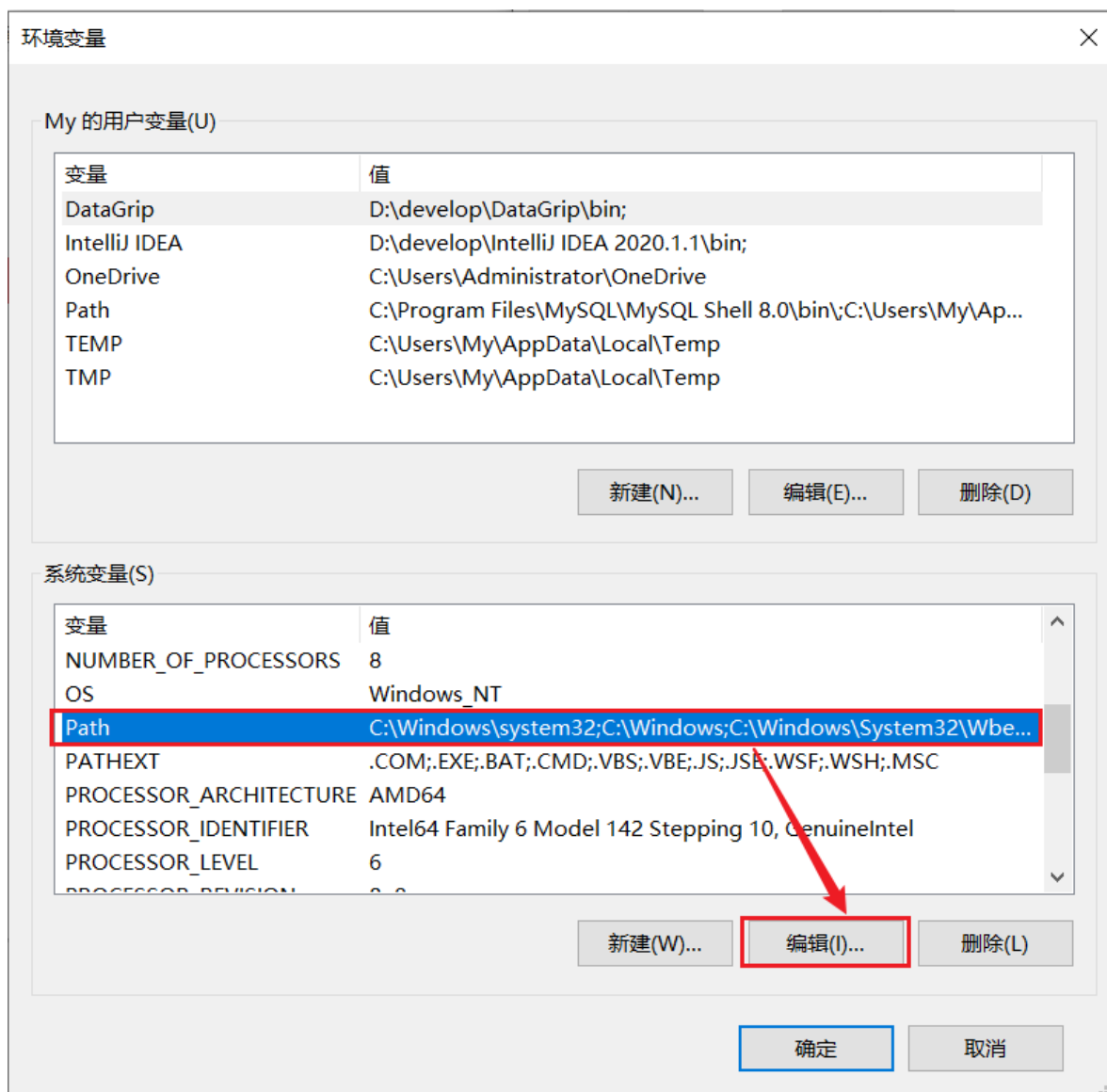
A. 在此电脑上，右键选择属性



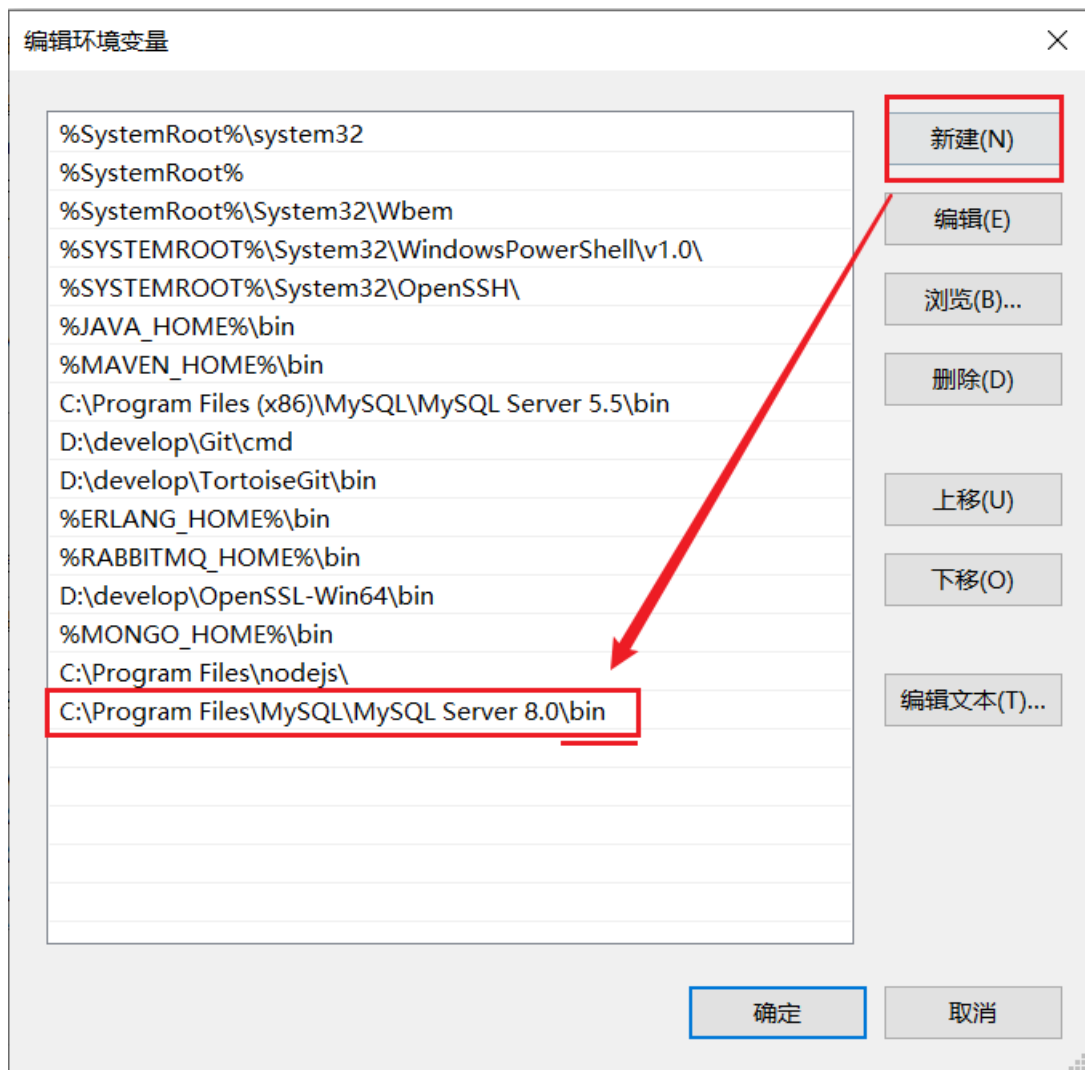
B. 点击左侧的 "高级系统设置", 选择环境变量



C. 找到 Path 系统变量，点击 "编辑"



D. 选择 "新建" , 将MySQL Server的安装目录下的bin目录添加到环境变量



1.2.4 启动停止

MySQL安装完成之后，在系统启动时，会自动启动MySQL服务，我们无需手动启动了。

当然，也可以手动的通过指令启动停止，以管理员身份运行cmd，进入命令行执行如下指令：

```
1 net start mysql80
2 net stop mysql80
```

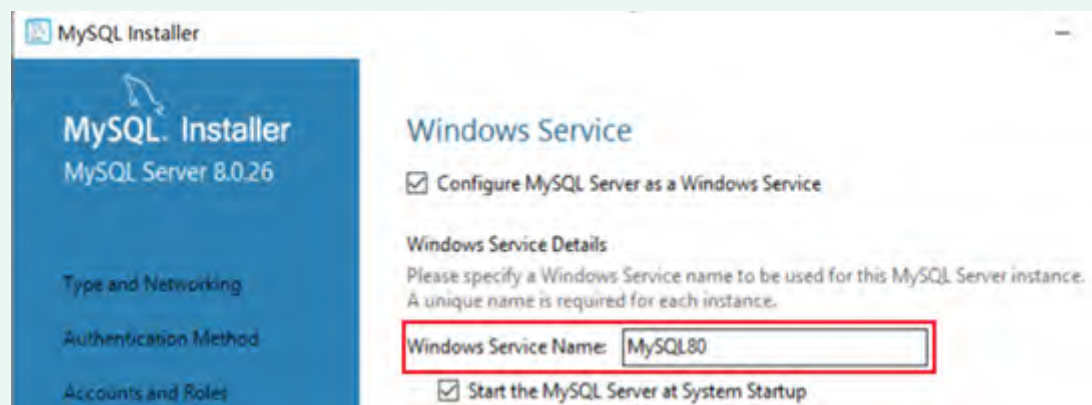
```
管理员: 命令提示符
Microsoft Windows [版本 10.0.17763.1577]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Windows\system32>net stop mysql80
MySQL80 服务正在停止。
MySQL80 服务已成功停止。

C:\Windows\system32>net start mysql80
MySQL80 服务正在启动。
MySQL80 服务已经启动成功。

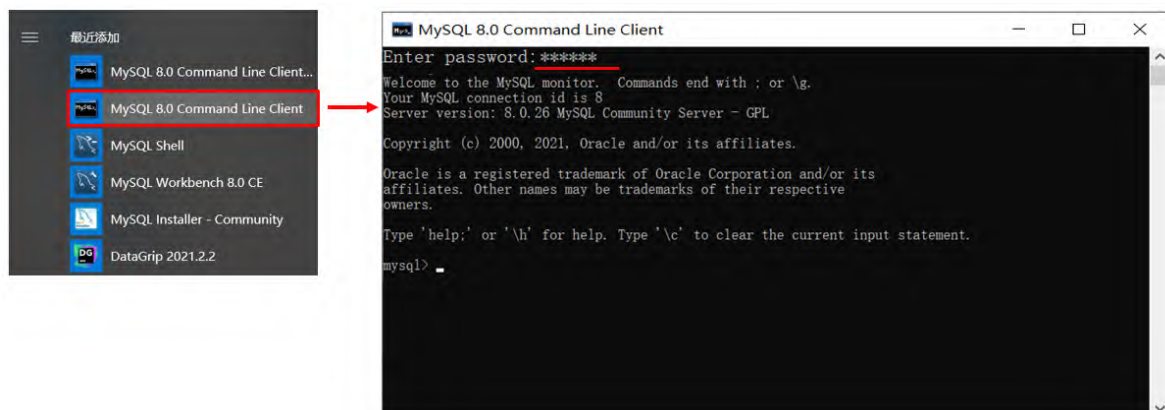
C:\Windows\system32>
```

注意：上述的 `mysql80` 是在我们安装MySQL时，默认指定的mysql的系统服务名，不是固定的，如果未改动，默认就是`mysql80`。



1.2.5 客户端连接

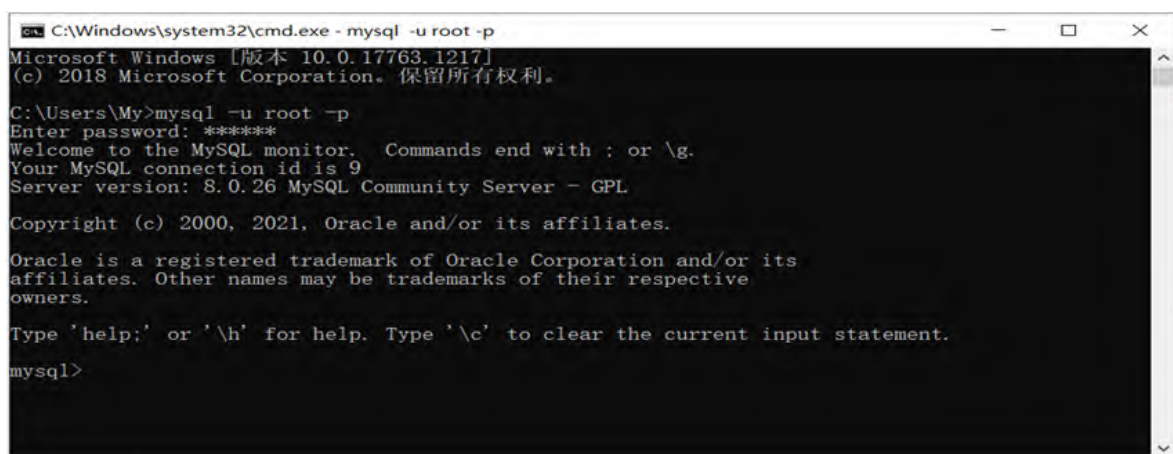
1). 方式一：使用MySQL提供的客户端命令行工具



2). 方式二：使用系统自带的命令行工具执行指令

```
1  mysql [-h 127.0.0.1] [-P 3306] -u root -p
2
3  参数:
4      -h : MySQL服务所在的主机IP
5      -P : MySQL服务端口号, 默认3306
6      -u : MySQL数据库用户名
7      -p : MySQL数据库用户名对应的密码
```

[] 内为可选参数，如果需要连接远程的MySQL，需要加上这两个参数来指定远程主机IP、端口，如果连接本地的MySQL，则无需指定这两个参数。



```
C:\Windows\system32\cmd.exe - mysql -u root -p
Microsoft Windows [版本 10.0.17763.1217]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\My>mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.26 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

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>
```

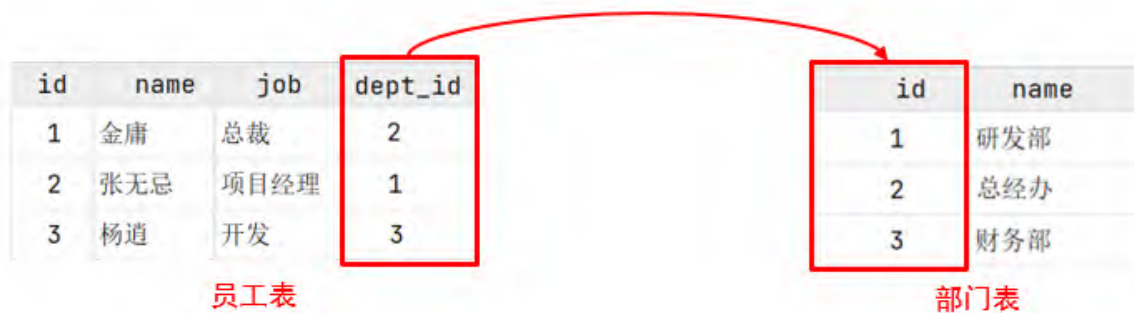
注意： 使用这种方式进行连接时，需要安装完毕后配置PATH环境变量。

1.2.6 数据模型

1). 关系型数据库 (RDBMS)

概念：建立在关系模型基础上，由多张相互连接的二维表组成的数据库。

而所谓二维表，指的是由行和列组成的表，如下图（就类似于Excel表格数据，有表头、有列、有行，还可以通过一列关联另外一个表格中的某一列数据）。我们之前提到的MySQL、Oracle、DB2、SQLServer这些都是属于关系型数据库，里面都是基于二维表存储数据的。简单说，基于二维表存储数据的数据库就成为关系型数据库，不是基于二维表存储数据的数据库，就是非关系型数据库。



特点:

- A. 使用表存储数据，格式统一，便于维护。
- B. 使用SQL语言操作，标准统一，使用方便。

2). 数据模型

MySQL是关系型数据库，是基于二维表进行数据存储的，具体的结构图下:



- 我们可以通过MySQL客户端连接数据库管理系统DBMS，然后通过DBMS操作数据库。
- 可以使用SQL语句，通过数据库管理系统操作数据库，以及操作数据库中的表结构及数据。
- 一个数据库服务器中可以创建多个数据库，一个数据库中也可以包含多张表，而一张表中又可以包含多行记录。

2. SQL

全称 Structured Query Language，结构化查询语言。操作关系型数据库的编程语言，定义了一套操作关系型数据库统一标准。

2.1 SQL通用语法

在学习具体的SQL语句之前，先来了解一下SQL语言的同于语法。

- 1) . SQL语句可以单行或多行书写，以分号结尾。
- 2) . SQL语句可以使用空格/缩进来增强语句的可读性。
- 3) . MySQL数据库的SQL语句不区分大小写，关键字建议使用大写。
- 4) . 注释：
 - 单行注释：-- 注释内容 或 # 注释内容
 - 多行注释：/* 注释内容 */

2.2 SQL分类

SQL语句，根据其功能，主要分为四类：DDL、DML、DQL、DCL。

分类	全称	说明
DDL	Data Definition Language	数据定义语言，用来定义数据库对象(数据库，表，字段)
DML	Data Manipulation Language	数据操作语言，用来对数据库表中的数据进行增删改
DQL	Data Query Language	数据查询语言，用来查询数据库中表的记录
DCL	Data Control Language	数据控制语言，用来创建数据库用户、控制数据库的访问权限

2.3 DDL

Data Definition Language，数据定义语言，用来定义数据库对象(数据库，表，字段)。

2.3.1 数据库操作

- 1) . 查询所有数据库

```
1 show databases ;
```

```
mysql>
mysql> show databases;
```

Database
information_schema
mysql
performance_schema
sys

```
4 rows in set (0.00 sec)
```

2). 查询当前数据库

```
1 select database();
```

3). 创建数据库

```
1 create database [ if not exists ] 数据库名 [ default charset 字符集 ] [ collate 排序规则 ] ;
```

案例:

A. 创建一个itcast数据库，使用数据库默认的字符集。

```
1 create database itcast;
```

```
mysql> create database itcast;
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> show databases;
```

Database
information_schema
itcast
mysql
performance_schema
sys

```
5 rows in set (0.00 sec)
```

在同一个数据库服务器中，不能创建两个名称相同的数据库，否则将会报错。

```
mysql> create database itcast;
ERROR 1007 (HY000): Can't create database 'itcast'; database exists
mysql>
```

可以通过if not exists 参数来解决这个问题，数据库不存在，则创建该数据库，如果存在，则不创建。

```
1 create database if not exists itcast;
```

```
mysql> create database if not exists itcast;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql>
mysql>
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| itcast      |
| mysql      |
| performance_schema |
| sys        |
+-----+
5 rows in set (0.00 sec)
```

B. 创建一个ittheima数据库，并且指定字符集

```
1 create database itheima default charset utf8mb4;
```

```
mysql> create database itheima default charset utf8mb4;
Query OK, 1 row affected (0.01 sec)

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

4). 删除数据库

```
1 drop database [ if exists ] 数据库名 ;
```

如果删除一个不存在的数据库，将会报错。此时，可以加上参数 `if exists`，如果数据库存在，再执行删除，否则不执行删除。

```
mysql>
mysql> drop database test;
ERROR 1008 (HY000): Can't drop database 'test'; database doesn't exist
mysql>
mysql>
mysql> drop database if exists test;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

5). 切换数据库

```
1 use 数据库名 ;
```

我们要操作某一个数据库下的表时，就需要通过该指令，切换到对应的数据库下，否则是不能操作的。比如，切换到 `itcast` 数据，执行如下SQL：

```
1 use itcast;
```

2.3.2 表操作

2.3.2.1 表操作-查询创建

1). 查询当前数据库所有表

```
1 show tables;
```

比如,我们可以切换到sys这个系统数据库,并查看系统数据库中的所有表结构。

```
1 use sys;  
2 show tables;
```

```
mysql>  
mysql> use sys;  
Database changed  
mysql>  
mysql> show tables;  
+-----+  
| Tables_in_sys |  
+-----+  
| host_summary |  
| host_summary_by_file_io |  
| host_summary_by_file_io_type |  
| host_summary_by_stage |  
| host_summary_by_statement_latency |  
| host_summary_by_statement_type |  
| innodb_buffer_stats_by_schema |  
| innodb_buffer_stats_by_table |  
| innodb_lock_waits |  
| io_by_thread_by_latency |  
| io_global_by_file_by_bytes |  
| io_global_by_file_by_latency |  
| io_global_by_wait_by_bytes |  
| io_global_by_wait_by_latency |  
| latest_file_io |  
| memory_by_host_by_current_bytes |  
| memory_by_thread_by_current_bytes |  
| memory_by_user_by_current_bytes |  
| memory_global_by_current_bytes |  
| memory_global_total |  
| metrics |  
| processlist |  
| ps_check_lost_instrumentation |  
| schema_auto_increment_columns |  
+-----+
```

2). 查看指定表结构

```
1 desc 表名 ;
```

通过这条指令,我们可以查看到指定表的字段,字段的类型、是否可以NULL,是否存在默认值等信息。

```
mysql> desc tb_user;  
+-----+  
| Field | Type | Null | Key | Default | Extra |  
+-----+  
| id | int | YES | | NULL | |  
| name | varchar(50) | YES | | NULL | |  
| age | int | YES | | NULL | |  
| gender | varchar(1) | YES | | NULL | |  
+-----+  
4 rows in set (0.01 sec)
```

3). 查询指定表的建表语句


```
1 show create table 表名 ;
```

通过这条指令，主要是用来查看建表语句的，而有部分参数我们在创建表的时候，并未指定也会查询到，因为这部分是数据库的默认值，如：存储引擎、字符集等。

```
mysql> show create table tb_user;
+-----+-----+
| Table | Create Table |
+-----+-----+
| tb_user | CREATE TABLE `tb_user` (
  `id` int DEFAULT NULL COMMENT '编号',
  `name` varchar(50) DEFAULT NULL COMMENT '姓名',
  `age` int DEFAULT NULL COMMENT '年龄',
  `gender` varchar(1) DEFAULT NULL COMMENT '性别'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci COMMENT='用户表' |
+-----+-----+
1 row in set (0.00 sec)
```

4). 创建表结构

```
1 CREATE TABLE 表名 (
2     字段1 字段1类型 [ COMMENT 字段1注释 ],
3     字段2 字段2类型 [ COMMENT 字段2注释 ],
4     字段3 字段3类型 [ COMMENT 字段3注释 ],
5     .....
6     字段n 字段n类型 [ COMMENT 字段n注释 ]
7 ) [ COMMENT 表注释 ] ;
```

注意：[...] 内为可选参数，最后一个字段后面没有逗号

比如，我们创建一张表 `tb_user`，对应的结构如下，那么建表语句为：

id	name	age	gender
1	令狐冲	28	男
2	风清扬	68	男
3	东方不败	32	男

```
1  create table tb_user(  
2      id int comment '编号',  
3      name varchar(50) comment '姓名',  
4      age int comment '年龄',  
5      gender varchar(1) comment '性别'  
6  ) comment '用户表';
```

2.3.2.2 表操作-数据类型

在上述的建表语句中，我们在指定字段的数据类型时，用到了int , varchar, 那么在MySQL中除了以上的数据类型，还有哪些常见的数据类型呢？ 接下来,我们就来详细介绍一下MySQL的数据类型。

MySQL中的数据类型有很多，主要分为三类：数值类型、字符串类型、日期时间类型。

1). 数值类型

类型	大小	有符号 (SIGNED) 范围	无符号 (UNSIGNED) 范围	描述
TINYINT	1byte	(-128, 127)	(0, 255)	小整数值
SMALLINT	2bytes	(-32768, 32767)	(0, 65535)	大整数值
MEDIUMINT	3bytes	(-8388608, 8388607)	(0, 16777215)	大整数值
INT/INTEGER	4bytes	(-2147483648, 2147483647)	(0, 4294967295)	大整数值
BIGINT	8bytes	(-2 ⁶³ , 2 ⁶³ -1)	(0, 2 ⁶⁴ -1)	极大整数值
FLOAT	4bytes	(-3.402823466 E+38, 3.402823466351 E+38)	0 和 (1.175494351 E-38, 3.402823466 E+38)	单精度浮点数值
DOUBLE	8bytes	(-1.7976931348623157 E+308, 1.7976931348623157 E+308)	0 和 (2.2250738585072014 E-308, 1.7976931348623157 E+308)	双精度浮点数值
DECIMAL		依赖于M (精度) 和D (标度) 的值	依赖于M (精度) 和D (标度) 的值	小数 (精确定点数)

1 如:

2 1). 年龄字段 -- 不会出现负数, 而且人的年龄不会太大

3 age tinyint unsigned

4

5 2). 分数 -- 总分100分, 最多出现一位小数

6 score double(4,1)

2). 字符串类型

类型	大小	描述
CHAR	0-255 bytes	定长字符串 (需要指定长度)
VARCHAR	0-65535 bytes	变长字符串 (需要指定长度)
TINYBLOB	0-255 bytes	不超过255个字符的二进制数据
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

char 与 varchar 都可以描述字符串, char是定长字符串, 指定长度多长, 就占用多少个字符, 和字段值的长度无关。而varchar是变长字符串, 指定的长度为最大占用长度。相对来说, char的性能会更高些。

```
1    如:
2
3    1). 用户名 username -----> 长度不定, 最长不会超过50
4
5    username varchar(50)
6
7
8    2). 性别 gender -----> 存储值, 不是男, 就是女
9
10   gender char(1)
11
12
13   3). 手机号 phone -----> 固定长度为11
14
15   phone char(11)
```

3). 日期时间类型

类型	大小	范围	格式	描述
DATE	3	1000-01-01 至 9999-12-31	YYYY-MM-DD	日期值
TIME	3	-838:59:59 至 838:59:59	HH:MM:SS	时间值或持续时间
YEAR	1	1901 至 2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00 至 9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:01 至 2038-01-19 03:14:07	YYYY-MM-DD HH:MM:SS	混合日期和时间值，时间戳

```
1  如：
2      1). 生日字段 birthday
3      birthday date
4
5      2). 创建时间 createtime
6      createtime datetime
```

2.3.2.3 表操作-案例

设计一张员工信息表，要求如下：

1. 编号（纯数字）
2. 员工工号（字符串类型，长度不超过10位）
3. 员工姓名（字符串类型，长度不超过10位）
4. 性别（男/女，存储一个汉字）
5. 年龄（正常人年龄，不可能存储负数）
6. 身份证号（二代身份证号均为18位，身份证中有x这样的字符）
7. 入职时间（取值年月日即可）

对应的建表语句如下：


```

1  create table emp(
2      id int comment '编号',
3      workno varchar(10) comment '工号',
4      name varchar(10) comment '姓名',
5      gender char(1) comment '性别',
6      age tinyint unsigned comment '年龄',
7      idcard char(18) comment '身份证号',
8      entrydate date comment '入职时间'
9  ) comment '员工表';

```

SQL语句编写完毕之后，就可以在MySQL的命令执行SQL，然后也可以通过 desc 指令查询表结构信息：

```
mysql> desc emp;
```

Field	Type	Null	Key	Default	Extra
id	int	YES		NULL	
workno	varchar(10)	YES		NULL	
name	varchar(10)	YES		NULL	
gender	char(1)	YES		NULL	
age	tinyint unsigned	YES		NULL	
idcard	char(18)	YES		NULL	
entrydate	date	YES		NULL	

7 rows in set (0.00 sec)

表结构创建好了，里面的name字段是varchar类型，最大长度为10，也就意味着如果超过10将会报错，如果我们想修改这个字段的类型 或 修改字段的长度该如何操作呢？接下来再来讲解DDL语句中，如何操作表字段。

2.3.2.4 表操作-修改

1). 添加字段

```
1  ALTER TABLE 表名 ADD 字段名 类型 (长度) [ COMMENT 注释 ] [ 约束 ];
```

案例：

为emp表增加一个新的字段“昵称”为nickname，类型为varchar(20)

```
1  ALTER TABLE emp ADD nickname varchar(20) COMMENT '昵称';
```

2). 修改数据类型

```
1  ALTER TABLE 表名 MODIFY 字段名 新数据类型 (长度);
```

3). 修改字段名和字段类型

```
1 ALTER TABLE 表名 CHANGE 旧字段名 新字段名 类型 (长度) [ COMMENT 注释 ] [ 约束 ];
```

案例:

将emp表的nickname字段修改为username, 类型为varchar(30)

```
1 ALTER TABLE emp CHANGE nickname username varchar(30) COMMENT '昵称';
```

4). 删除字段

```
1 ALTER TABLE 表名 DROP 字段名;
```

案例:

将emp表的字段username删除

```
1 ALTER TABLE emp DROP username;
```

5). 修改表名

```
1 ALTER TABLE 表名 RENAME TO 新表名;
```

案例:

将emp表的表名修改为 employee

```
1 ALTER TABLE emp RENAME TO employee;
```

2.3.2.5 表操作-删除

1). 删除表

```
1 DROP TABLE [ IF EXISTS ] 表名;
```

可选项 IF EXISTS 代表, 只有表名存在时才会删除该表, 表名不存在, 则不执行删除操作(如果不加该参数项, 删除一张不存在的表, 执行将会报错)。

案例:

如果tb_user表存在, 则删除tb_user表

```
1 DROP TABLE IF EXISTS tb_user;
```

2). 删除指定表，并重新创建表

```
1 TRUNCATE TABLE 表名;
```

注意：在删除表的时候，表中的全部数据也都会被删除。

2.4 图形化界面工具

上述，我们已经讲解了通过DDL语句，如何操作数据库、操作表、操作表中的字段，而通过DDL语句执行在命令进行操作，主要存在以下两点问题：

- 1). 会影响开发效率 ；
- 2). 使用起来，并不直观，并不方便 ；

所以呢，我们在日常的开发中，会借助于MySQL的图形化界面，来简化开发，提高开发效率。而目前mysql主流的图形化界面工具，有以下几种：



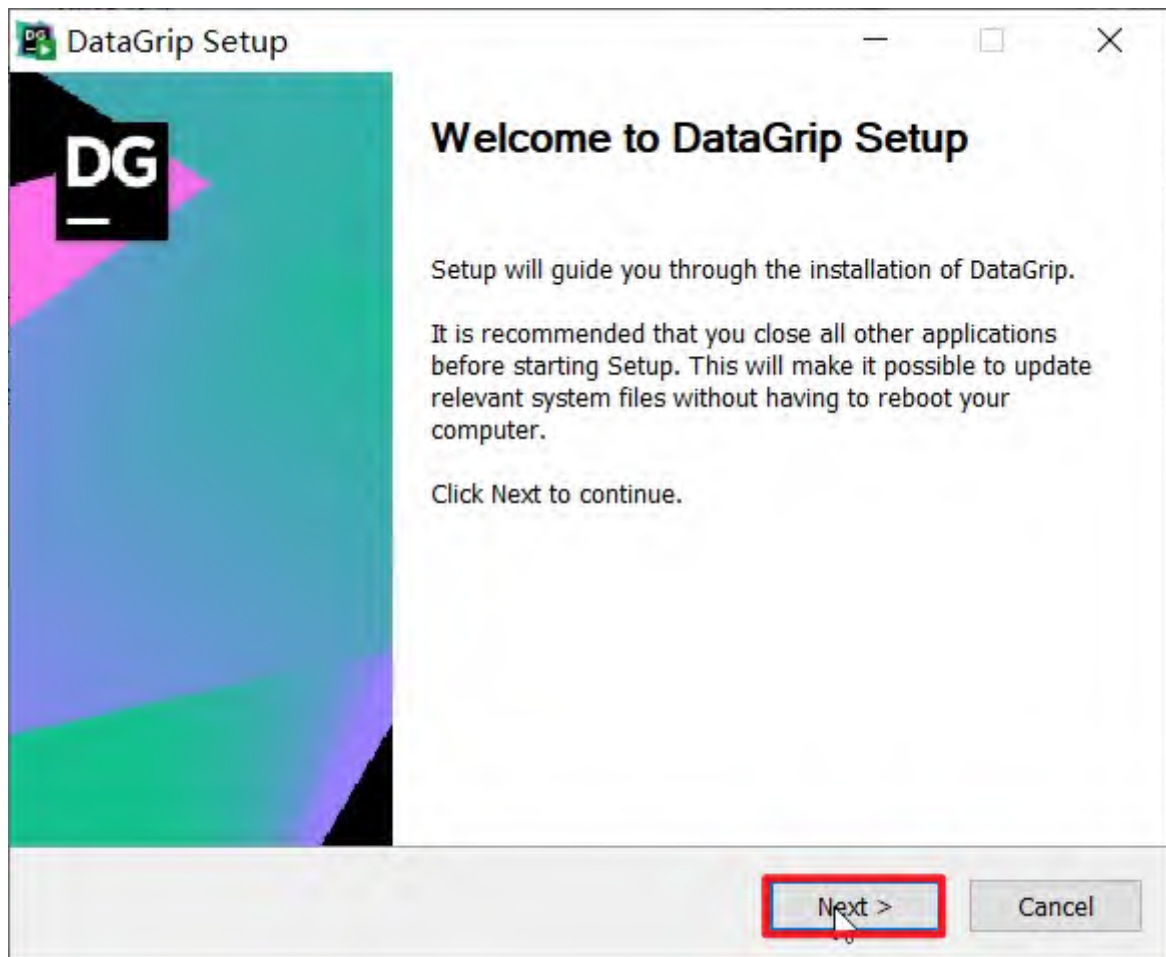
而本次课程中，选择最后一种DataGrip，这种图形化界面工具，功能更加强大，界面提示更加友好，是我们使用MySQL的不二之选。接下来，我们来介绍一下DataGrip该如何安装、使用。

2.4.1 安装

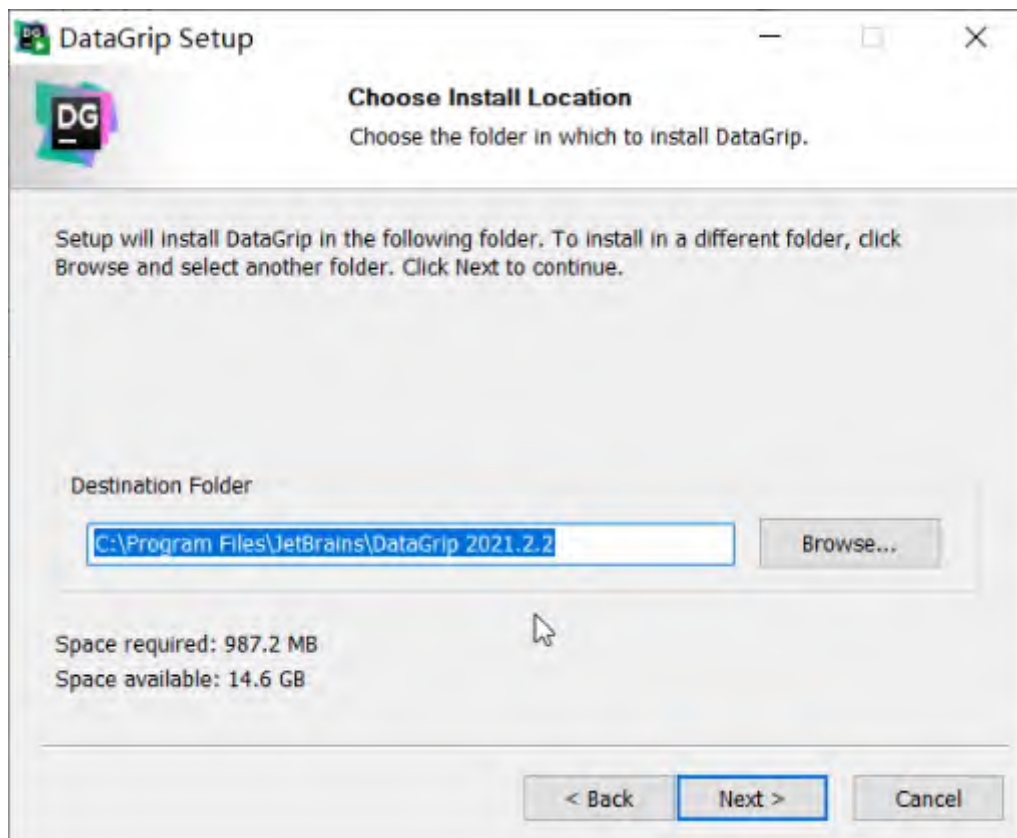
- 1). 找到资料中准备好的安装包，双击开始安装

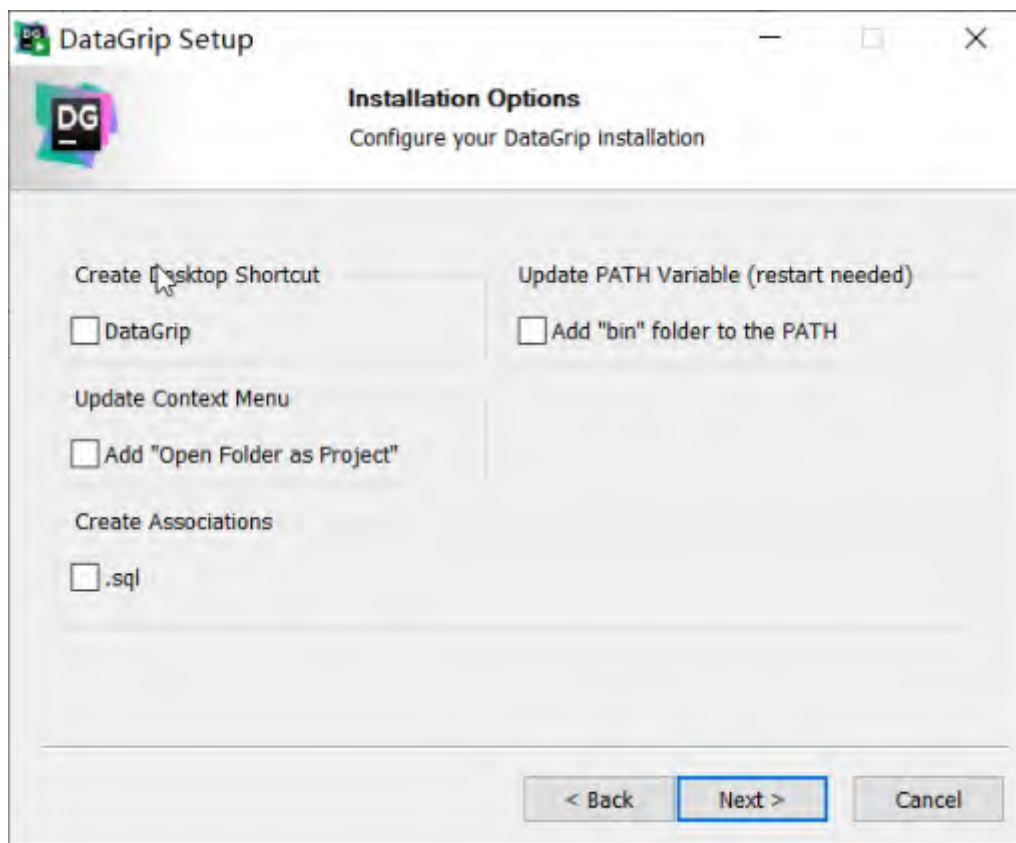


2). 点击next, 一步一步的完成安装

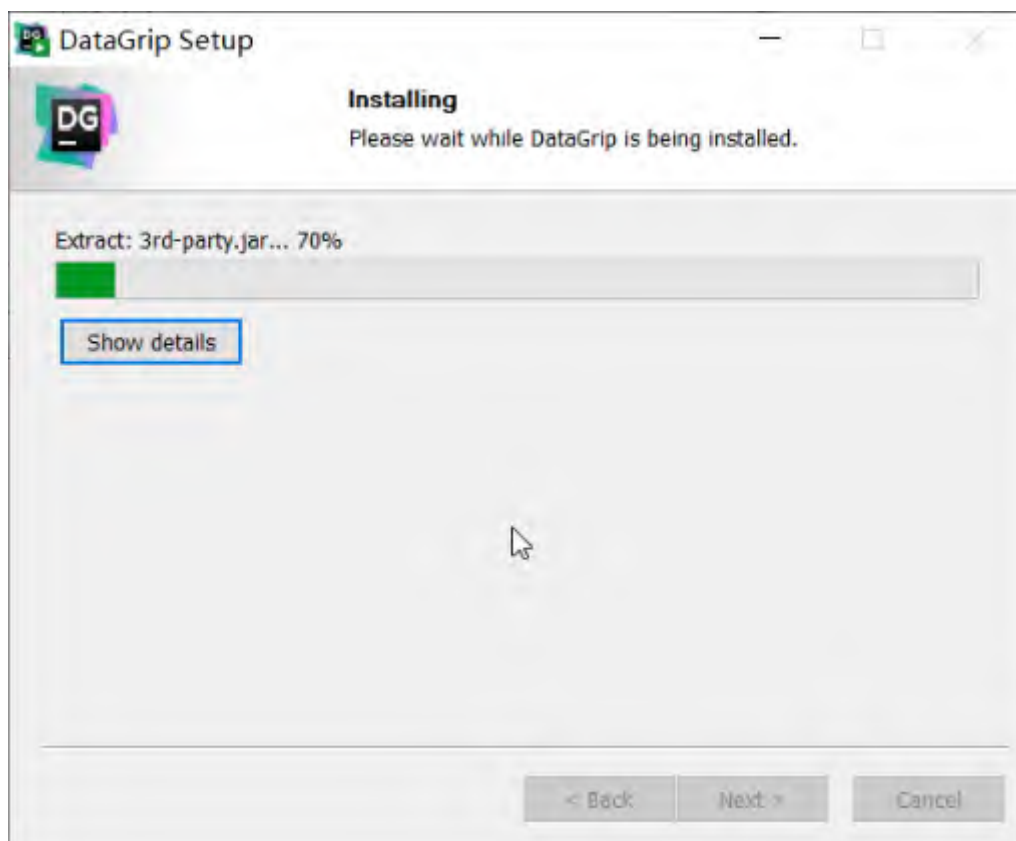


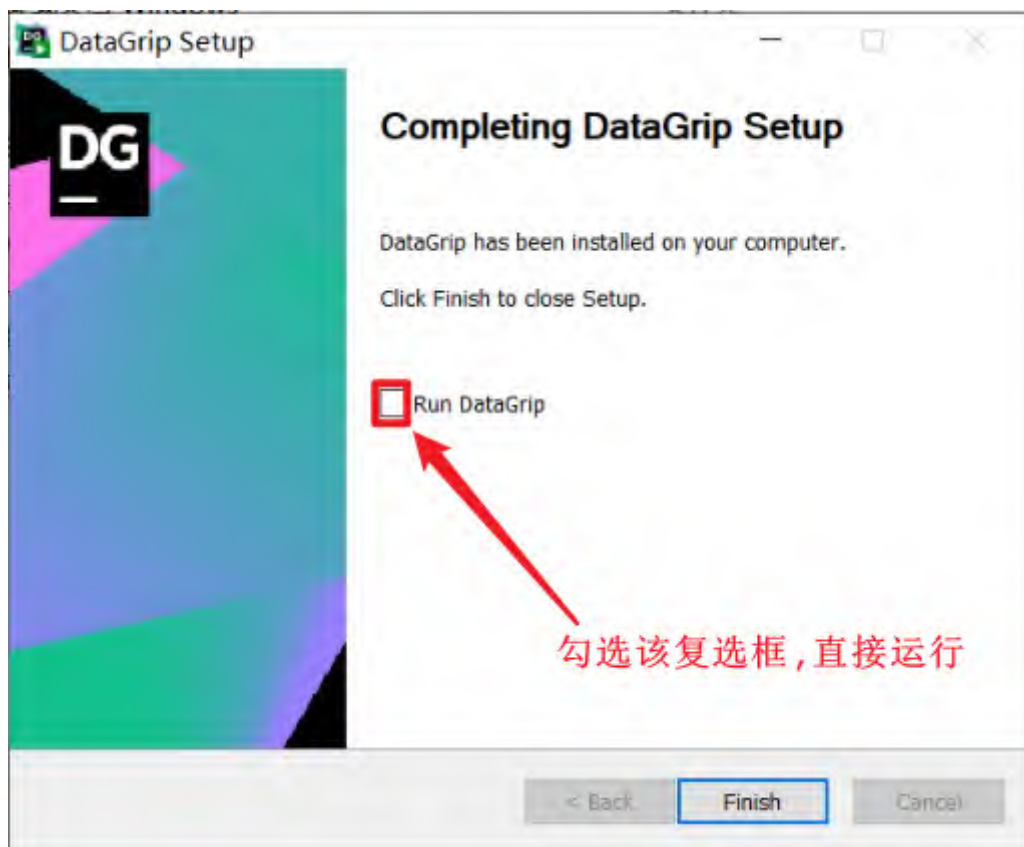
选择DataGrip的安装目录, 然后选择下一步





下一步，执行安装

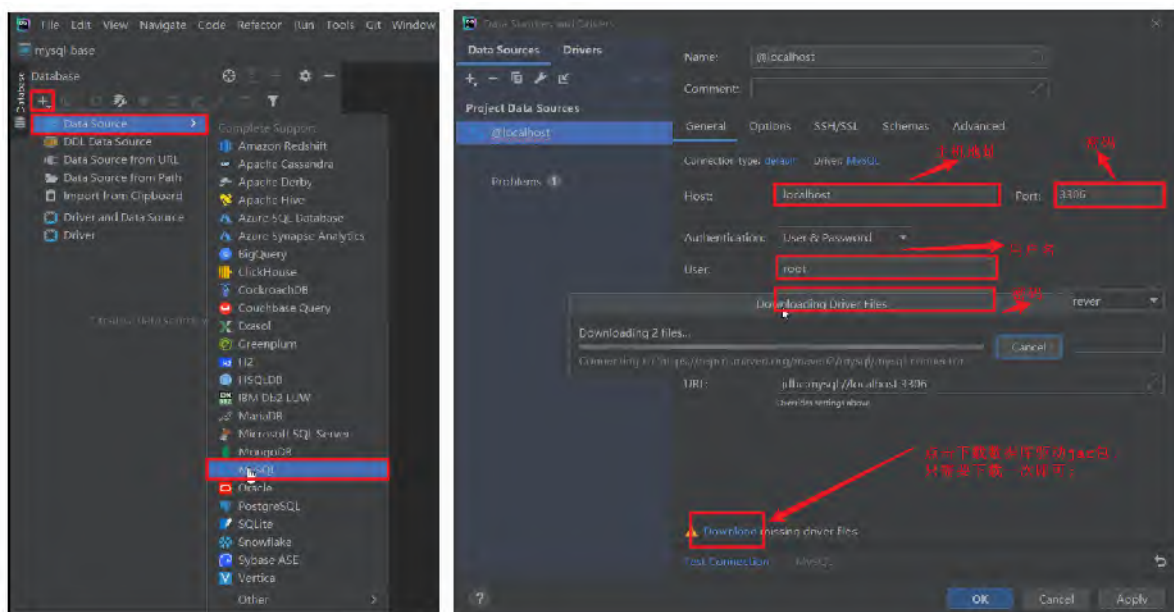




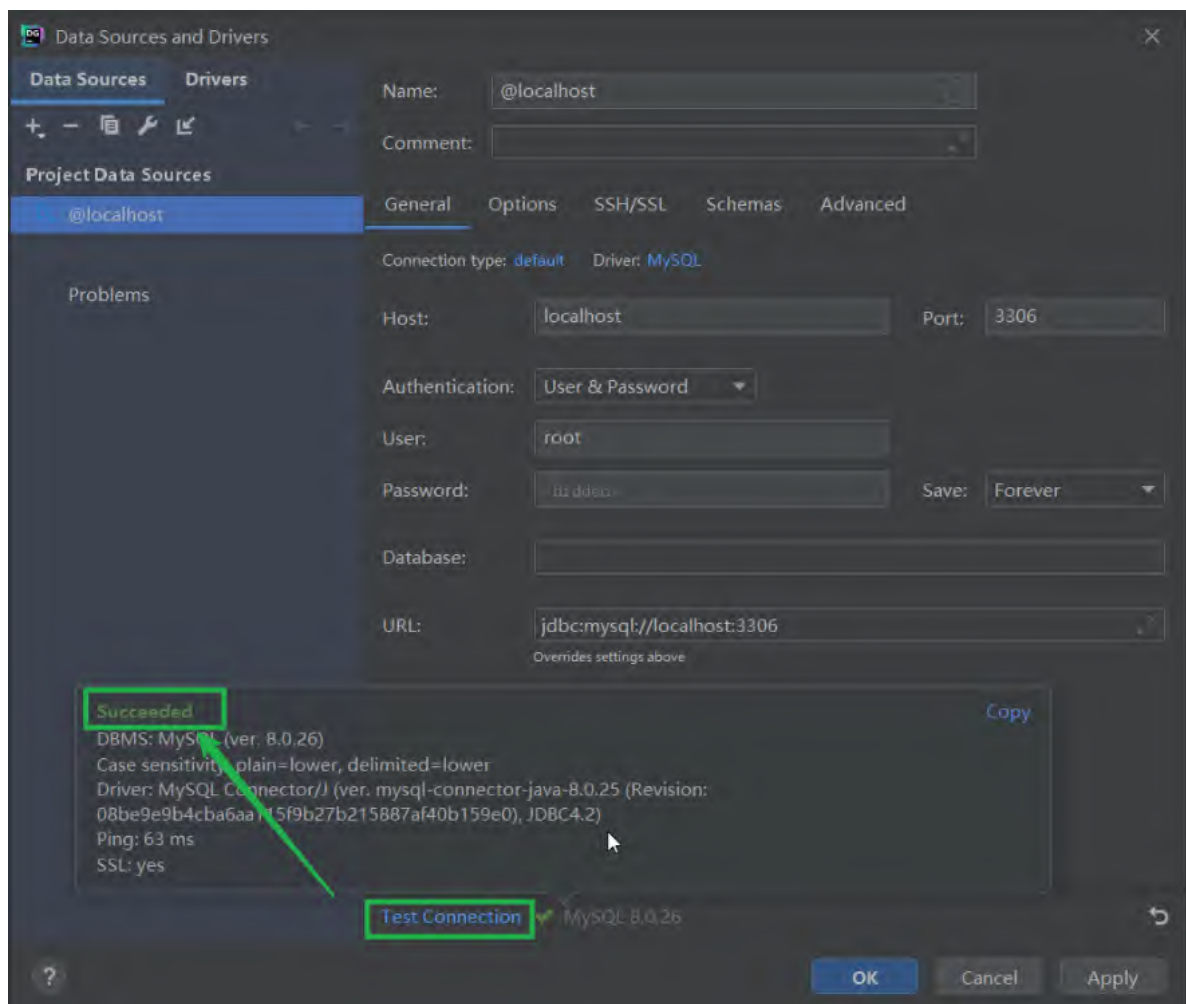
2.4.2 使用

1). 添加数据源

参考图示，一步步操作即可

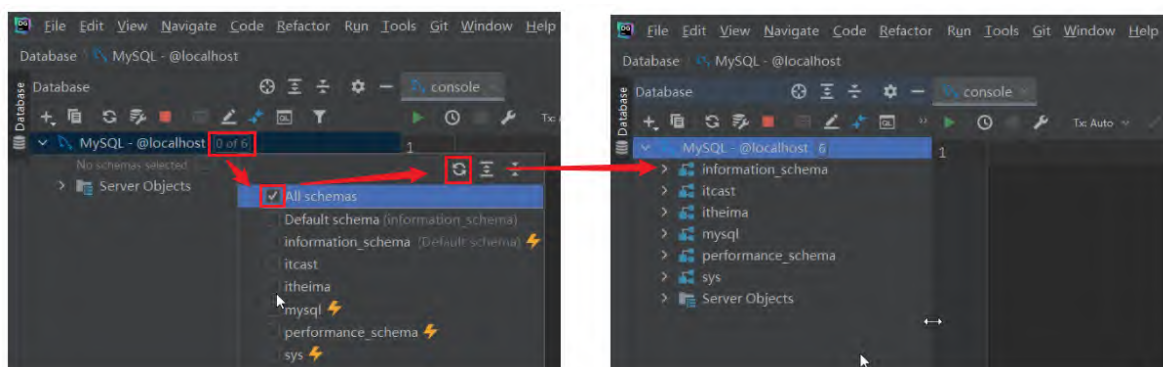


配置以及驱动jar包下载完毕之后，就可以点击 "Test Connection" 就可以测试，是否可以连接 MySQL，如果出现 "Succeeded"，就表明连接成功了。

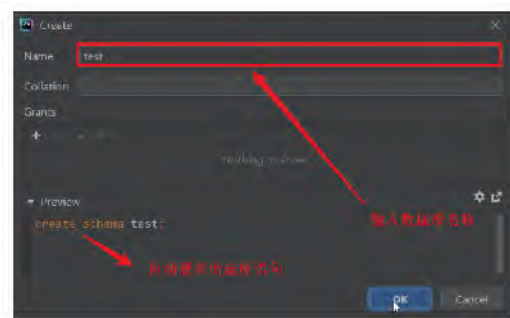
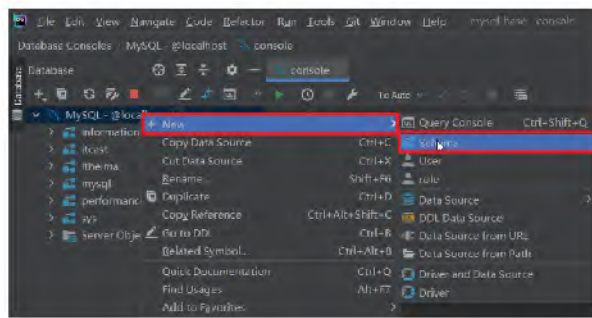


2). 展示所有数据库

连接上了MySQL服务之后，并未展示出所有的数据库，此时，我们需要设置，展示所有的数据库，具体操作如下：



3). 创建数据库



注意：

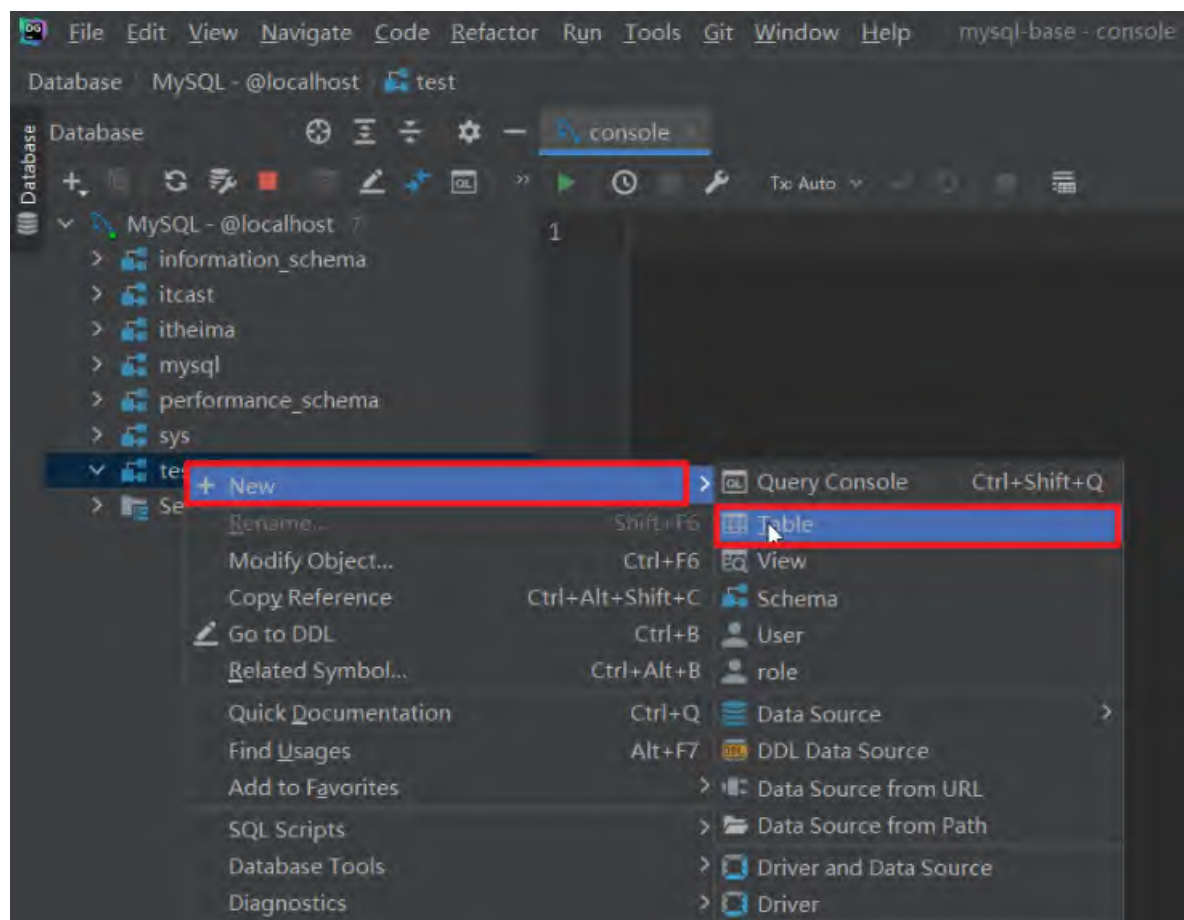
以下两种方式都可以创建数据库：

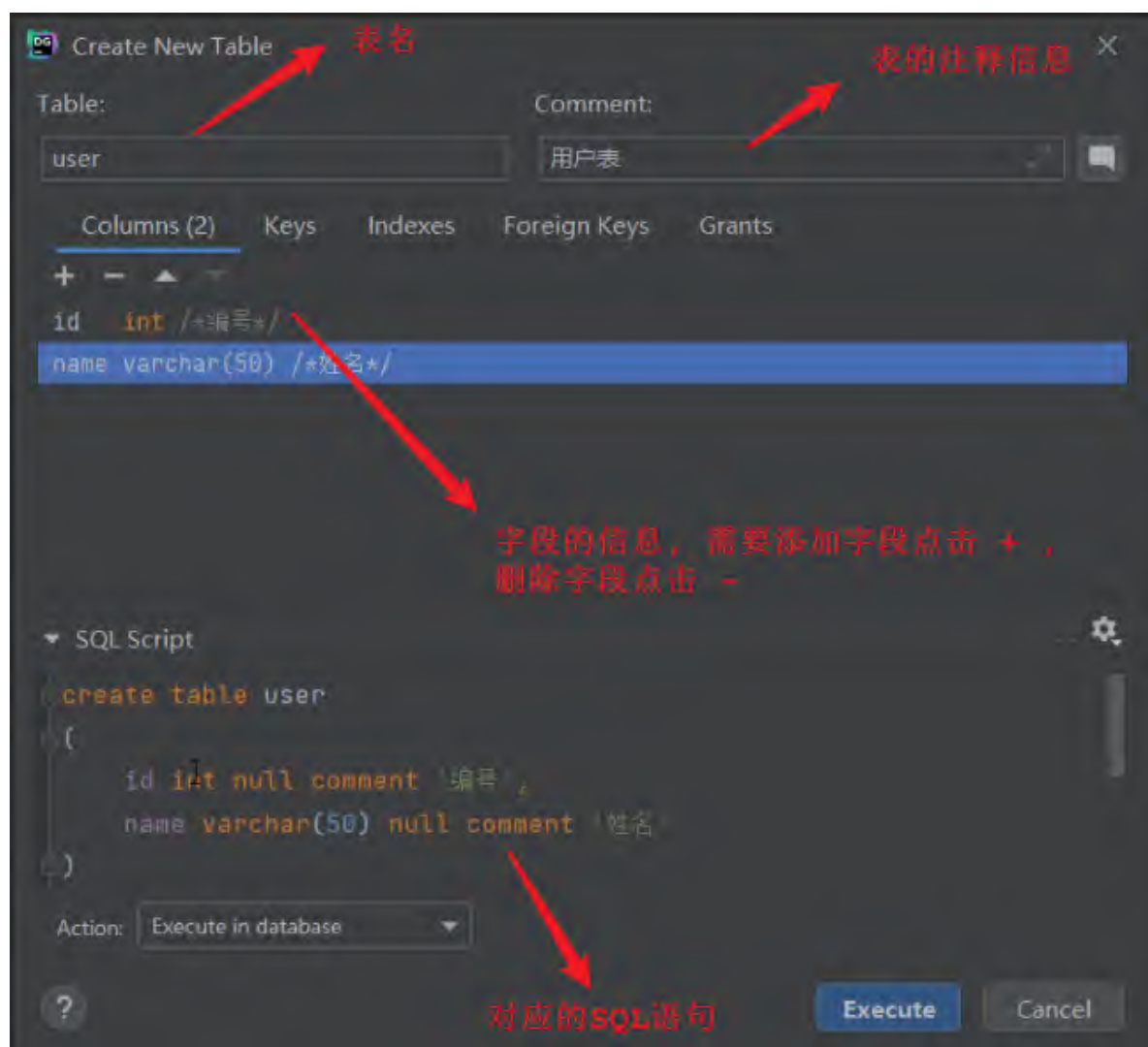
A. `create database db01;`

B. `create schema db01;`

4). 创建表

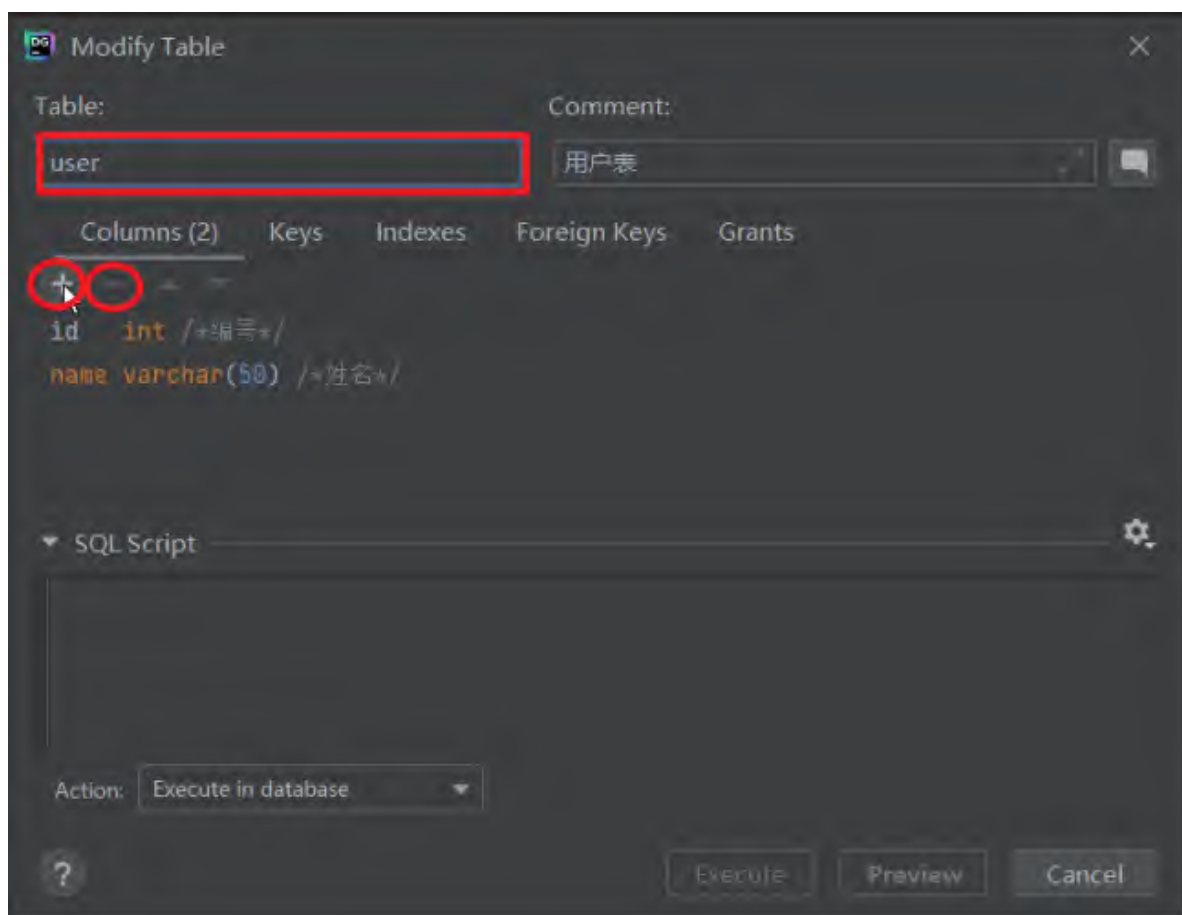
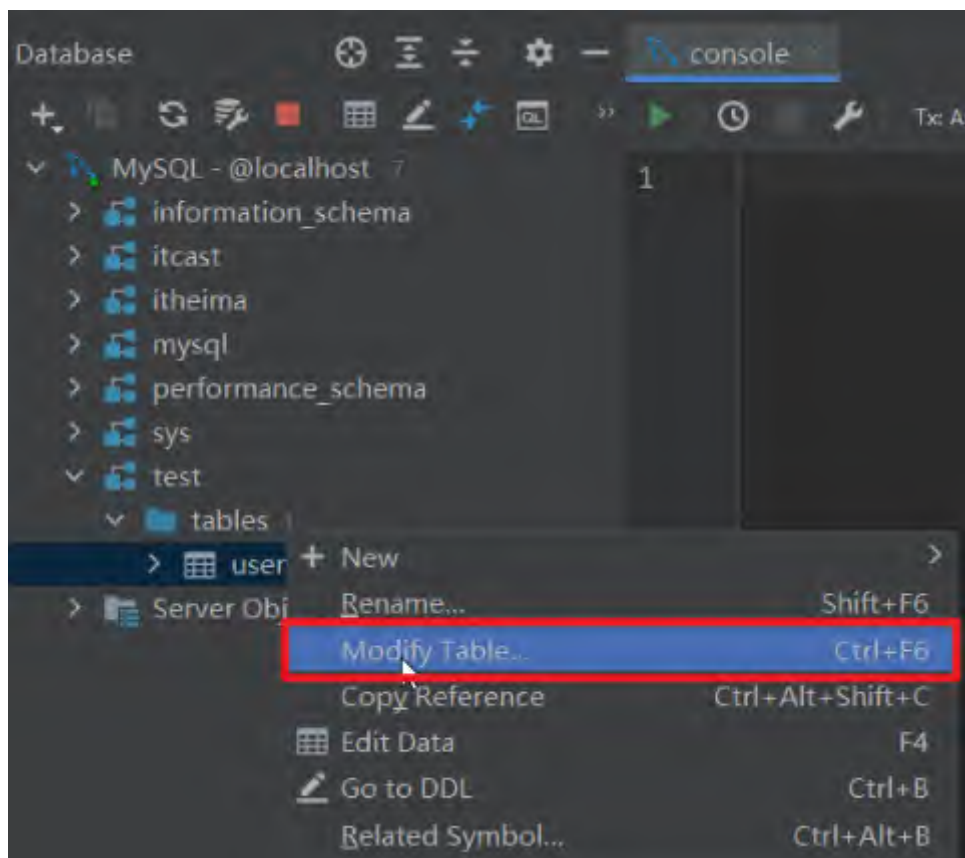
在指定的数据库上面右键，选择new --> Table





5). 修改表结构

在需要修改的表上, 右键选择 "Modify Table..."



如果想增加字段，直接点击+号，录入字段信息，然后点击Execute即可。

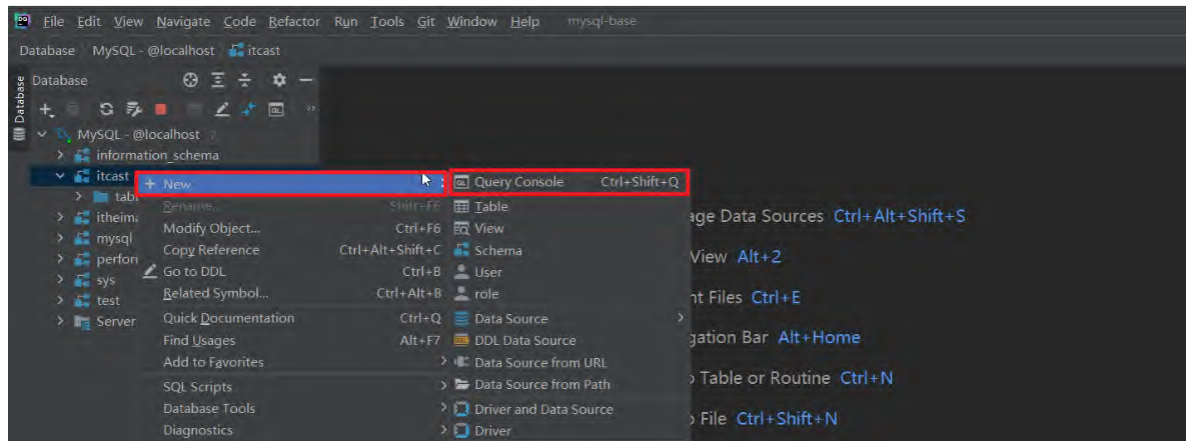
如果想删除字段，直接点击-号，就可以删除字段，然后点击Execute即可。

如果想修改字段，双击对应的字段，修改字段信息，然后点击Execute即可。

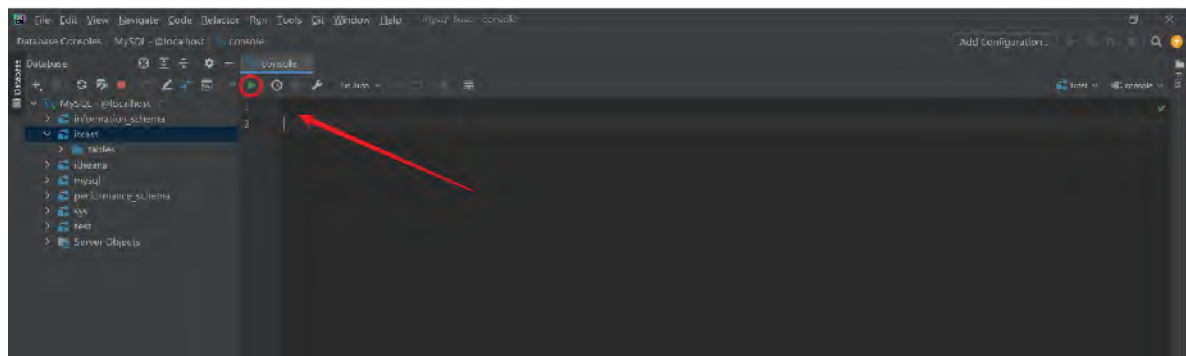
如果要修改表名，或表的注释，直接在输入框修改，然后点击Execute即可。

6). 在DataGrip中执行SQL语句

在指定的数据库上，右键，选择 New --> Query Console



然后就可以在打开的Query Console控制台，并在控制台中编写SQL，执行SQL。



2.5 DML

DML英文全称是Data Manipulation Language (数据操作语言)，用来对数据库中表的数据记录进行增、删、改操作。

- 添加数据 (INSERT)
- 修改数据 (UPDATE)
- 删除数据 (DELETE)

2.5.1 添加数据

1). 给指定字段添加数据

```
1  INSERT INTO 表名 (字段名1, 字段名2, ...) VALUES (值1, 值2, ...);
```

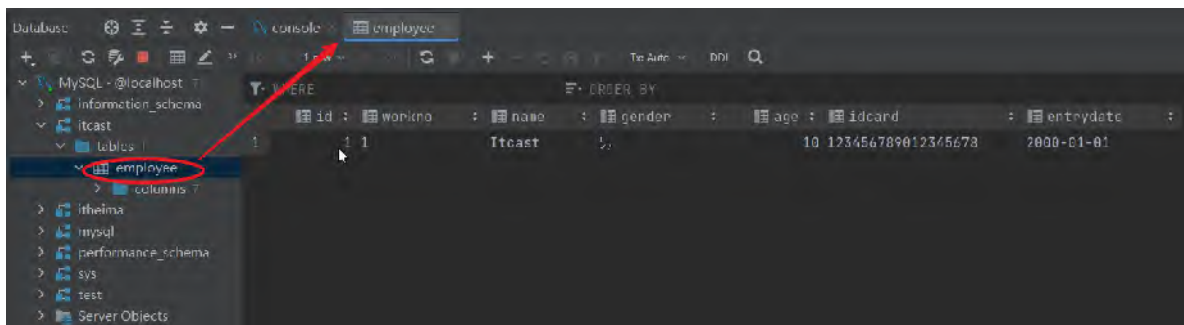
案例：给employee表所有的字段添加数据；

```
1  insert into employee(id,workno,name,gender,age,idcard,entrydate)
    values(1,'1','Itcast','男',10,'123456789012345678','2000-01-01');
```

插入数据完成之后，我们有两种方式，查询数据库的数据：

A. 方式一

在左侧的表名上双击，就可以查看这张表的数据。



B. 方式二

可以直接一条查询数据的SQL语句，语句如下：

```
1  select * from employee;
```

案例：给employee表所有的字段添加数据

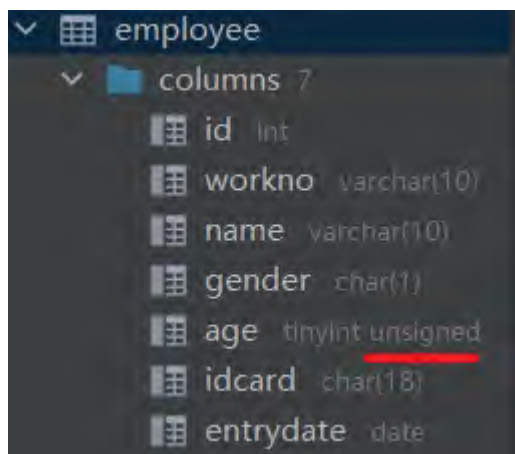
执行如下SQL，添加的年龄字段值为-1。

```
1  insert into employee(id,workno,name,gender,age,idcard,entrydate)
    values(1,'1','Itcast','男',-1,'123456789012345678','2000-01-01');
```

执行上述的SQL语句时，报错了，具体的错误信息如下：

```
[22001][1264] Data truncation: Out of range value for column 'age' at row 1
```

因为 employee 表的age字段类型为 tinyint，而且还是无符号的 unsigned，所以取值只能在 0-255 之间。



2). 给全部字段添加数据

```
1  INSERT INTO 表名 VALUES (值1, 值2, ...);
```

案例：插入数据到employee表，具体的SQL如下：

```
1  insert into employee values(2,'2','张无忌','男',18,'123456789012345670','2005-01-01');
```

3). 批量添加数据

```
1  INSERT INTO 表名 (字段名1, 字段名2, ...) VALUES (值1, 值2, ...), (值1, 值2, ...), (值1, 值2, ...);
```

```
1  INSERT INTO 表名 VALUES (值1, 值2, ...), (值1, 值2, ...), (值1, 值2, ...);
```

案例：批量插入数据到employee表，具体的SQL如下：

```
1  insert into employee values(3,'3','韦一笑','男',38,'123456789012345670','2005-01-01'), (4,'4','赵敏','女',18,'123456789012345670','2005-01-01');
```

注意事项：

- 插入数据时，指定的字段顺序需要与值的顺序是一一对应的。

- 字符串和日期型数据应该包含在引号中。
- 插入的数据大小，应该在字段的规定范围内。

2.5.2 修改数据

修改数据的具体语法为：

```
1  UPDATE    表名    SET    字段名1 = 值1 , 字段名2 = 值2 , .... [ WHERE 条件 ] ;
```

案例：

A. 修改id为1的数据，将name修改为itheima

```
1  update employee set name = 'itheima' where id = 1;
```

B. 修改id为1的数据，将name修改为小昭，gender修改为 女

```
1  update employee set name = '小昭' , gender = '女' where id = 1;
```

C. 将所有的员工入职日期修改为 2008-01-01

```
1  update employee set entrydate = '2008-01-01';
```

注意事项：

修改语句的条件可以有，也可以没有，如果没有条件，则会修改整张表的所有数据。

2.5.3 删除数据

删除数据的具体语法为：

```
1  DELETE FROM 表名 [ WHERE 条件 ] ;
```

案例：

A. 删除gender为女的员工

```
1 delete from employee where gender = '女';
```

B. 删除所有员工

```
1 delete from employee;
```

注意事项:

- DELETE 语句的条件可以有,也可以没有,如果没有条件,则会删除整张表的所有数据。
- DELETE 语句不能删除某一个字段的值(可以使用UPDATE,将该字段值置为NULL即可)。
- 当进行删除全部数据操作时, datagrip会提示我们,询问是否确认删除,我们直接点击Execute即可。

Unsafe query: 'Delete' statement without 'where' clears all data in the table

Execute Execute and Suppress

2.6 DQL

DQL英文全称是Data Query Language(数据查询语言), 数据查询语言, 用来查询数据库中表的记录。

查询关键字: SELECT

在一个正常的业务系统中, 查询操作的频次是要远高于增删改的, 当我们去访问企业官网、电商网站, 在这些网站中所看到的数据, 实际都是需要从数据库中查询并展示的。而且在查询的过程中, 可能还会涉及到条件、排序、分页等操作。



那么，本小节我们主要学习的就是如何进行数据的查询操作。 我们先来完成如下数据准备工作：

```

1  drop table if exists employee;

2

3  create table emp(

4      id int comment '编号',

5      workno varchar(10) comment '工号',

6      name varchar(10) comment '姓名',

7      gender char(1) comment '性别',

8      age tinyint unsigned comment '年龄',

9      idcard char(18) comment '身份证号',

10     workaddress varchar(50) comment '工作地址',

11     entrydate date comment '入职时间'

12 )comment '员工表';

13

14 INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)

VALUES (1, '00001', '柳岩666', '女', 20, '123456789012345678', '北京', '2000-01-

01');

15 INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)

VALUES (2, '00002', '张无忌', '男', 18, '123456789012345670', '北京', '2005-09-

01');

```

```
16  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (3, '00003', '韦一笑', '男', 38, '123456789712345670', '上海', '2005-08-
01');

17  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (4, '00004', '赵敏', '女', 18, '123456757123845670', '北京', '2009-12-01');

18  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (5, '00005', '小昭', '女', 16, '123456769012345678', '上海', '2007-07-01');

19  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (6, '00006', '杨逍', '男', 28, '12345678931234567X', '北京', '2006-01-01');

20  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (7, '00007', '范瑶', '男', 40, '123456789212345670', '北京', '2005-05-01');

21  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (8, '00008', '黛绮丝', '女', 38, '123456157123645670', '天津', '2015-05-
01');

22  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (9, '00009', '范凉凉', '女', 45, '123156789012345678', '北京', '2010-04-
01');

23  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (10, '00010', '陈友谅', '男', 53, '123456789012345670', '上海', '2011-01-
01');

24  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (11, '00011', '张士诚', '男', 55, '123567897123465670', '江苏', '2015-05-
01');

25  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (12, '00012', '常遇春', '男', 32, '123446757152345670', '北京', '2004-02-
01');

26  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (13, '00013', '张三丰', '男', 88, '123656789012345678', '江苏', '2020-11-
01');

27  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (14, '00014', '灭绝', '女', 65, '123456719012345670', '西安', '2019-05-
01');

28  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (15, '00015', '胡青牛', '男', 70, '12345674971234567X', '西安', '2018-04-
01');

29  INSERT INTO emp (id, workno, name, gender, age, idcard, workaddress, entrydate)
VALUES (16, '00016', '周芷若', '女', 18, null, '北京', '2012-06-01');
```

准备完毕后，我们就可以看到emp表中准备的16条数据。接下来，我们再来完成DQL语法的学习。

2.6.1 基本语法

DQL 查询语句，语法结构如下：

```
1  SELECT
2      字段列表
3  FROM
4      表名列表
5  WHERE
6      条件列表
7  GROUP BY
8      分组字段列表
9  HAVING
10     分组后条件列表
11 ORDER BY
12     排序字段列表
13 LIMIT
14     分页参数
```

我们在讲解这部分内容的时候，会将上面的完整语法进行拆分，分为以下几个部分：

- 基本查询（不带任何条件）
- 条件查询（WHERE）
- 聚合函数（count、max、min、avg、sum）
- 分组查询（group by）
- 排序查询（order by）
- 分页查询（limit）

2.6.2 基础查询

在基本查询的DQL语句中，不带任何的查询条件，查询的语法如下：

1). 查询多个字段

```
1  SELECT  字段1, 字段2, 字段3 ... FROM  表名 ;
```

```
1  SELECT  * FROM  表名 ;
```

注意：* 号代表查询所有字段，在实际开发中尽量少用（不直观、影响效率）。

2). 字段设置别名

```
1 SELECT  字段1  [ AS  别名1 ] , 字段2  [ AS  别名2 ]  ... FROM  表名;
```

```
1 SELECT  字段1  [ 别名1 ] , 字段2  [ 别名2 ]  ... FROM  表名;
```

3). 去除重复记录

```
1 SELECT  DISTINCT  字段列表 FROM  表名;
```

案例：

A. 查询指定字段 name, workno, age并返回

```
1 select name,workno,age from emp;
```

B. 查询返回所有字段

```
1 select id ,workno,name,gender,age,idcard,workaddress,entrydate from emp;
```

```
1 select * from emp;
```

C. 查询所有员工的工作地址,起别名

```
1 select workaddress as '工作地址' from emp;
```

```
1 -- as可以省略
```

```
2 select workaddress '工作地址' from emp;
```

D. 查询公司员工的上班地址有哪些(不要重复)

```
1 select distinct workaddress '工作地址' from emp;
```

2.6.3 条件查询

1). 语法


```
1 SELECT  字段列表 FROM  表名 WHERE  条件列表 ;
```

2) . 条件

常用的比较运算符如下：

比较运算符	功能
>	大于
>=	大于等于
<	小于
<=	小于等于
=	等于
<> 或 !=	不等于
BETWEEN ... AND ...	在某个范围之内(含最小、最大值)
IN(...)	在in之后的列表中的值，多选一
LIKE 占位符	模糊匹配(_匹配单个字符，%匹配任意个字符)
IS NULL	是NULL

常用的逻辑运算符如下：

逻辑运算符	功能
AND 或 &&	并且（多个条件同时成立）
OR 或	或者（多个条件任意一个成立）
NOT 或 !	非，不是

案例：

A. 查询年龄等于 88 的员工

```
1 select * from emp where age = 88;
```

B. 查询年龄小于 20 的员工信息

```
1 select * from emp where age < 20;
```

C. 查询年龄小于等于 20 的员工信息

```
1 select * from emp where age <= 20;
```

D. 查询没有身份证号的员工信息

```
1 select * from emp where idcard is null;
```

E. 查询有身份证号的员工信息

```
1 select * from emp where idcard is not null;
```

F. 查询年龄不等于 88 的员工信息

```
1 select * from emp where age != 88;
```

```
2 select * from emp where age <> 88;
```

G. 查询年龄在15岁(包含) 到 20岁(包含)之间的员工信息

```
1 select * from emp where age >= 15 && age <= 20;
```

```
2 select * from emp where age >= 15 and age <= 20;
```

```
3 select * from emp where age between 15 and 20;
```

H. 查询性别为 女 且年龄小于 25岁的员工信息

```
1 select * from emp where gender = '女' and age < 25;
```

I. 查询年龄等于18 或 20 或 40 的员工信息

```
1 select * from emp where age = 18 or age = 20 or age = 40;
```

```
2 select * from emp where age in(18,20,40);
```

J. 查询姓名为两个字的员工信息 _ %

```
1 select * from emp where name like '__';
```

K. 查询身份证号最后一位是X的员工信息

```
1 select * from emp where idcard like '%X';
```

```
2 select * from emp where idcard like '_____X';
```

2.6.4 聚合函数

1). 介绍

将一系列数据作为一个整体，进行纵向计算。

2). 常见的聚合函数

函数	功能
count	统计数量
max	最大值
min	最小值
avg	平均值
sum	求和

3). 语法

```
1  SELECT  聚合函数(字段列表) FROM  表名 ;
```

注意：NULL值是不参与所有聚合函数运算的。

案例：

A. 统计该企业员工数量

```
1  select count(*) from emp; -- 统计的是总记录数
2  select count(idcard) from emp; -- 统计的是idcard字段不为null的记录数
```

对于count聚合函数，统计符合条件的总记录数，还可以通过 count(数字/字符串) 的形式进行统计查询，比如：

```
1  select count(1) from emp;
```

对于count(*)、count(字段)、count(1) 的具体原理，我们在进阶篇中SQL优化部分会详细讲解，此处大家只需要知道如何使用即可。

B. 统计该企业员工的平均年龄

```
1  select avg(age) from emp;
```

C. 统计该企业员工的最大年龄

```
1 select max(age) from emp;
```

D. 统计该企业员工的最小年龄

```
1 select min(age) from emp;
```

E. 统计西安地区员工的年龄之和

```
1 select sum(age) from emp where workaddress = '西安';
```

2.6.5 分组查询

1). 语法

```
1 SELECT  字段列表 FROM  表名 [ WHERE  条件 ] GROUP BY  分组字段名 [ HAVING  分组后过滤条件 ];
```

2). where与having区别

- 执行时机不同：where是分组之前进行过滤，不满足where条件，不参与分组；而having是分组之后对结果进行过滤。
- 判断条件不同：where不能对聚合函数进行判断，而having可以。

注意事项：

- 分组之后，查询的字段一般为聚合函数和分组字段，查询其他字段无任何意义。
- 执行顺序：where > 聚合函数 > having 。
- 支持多字段分组，具体语法为：group by columnA,columnB

案例：

A. 根据性别分组，统计男性员工和女性员工的数量

```
1 select gender, count(*) from emp group by gender ;
```

B. 根据性别分组，统计男性员工和女性员工的平均年龄

```
1 select gender, avg(age) from emp group by gender ;
```

C. 查询年龄小于45的员工，并根据工作地址分组，获取员工数量大于等于3的工作地址

```
1 select workaddress, count(*) address_count from emp where age < 45 group by  
workaddress having address_count >= 3;
```

D. 统计各个工作地址上班的男性及女性员工的数量

```
1 select workaddress, gender, count(*) '数量' from emp group by gender, workaddress  
;
```

2.6.6 排序查询

排序在日常开发中是非常常见的一个操作，有升序排序，也有降序排序。



1). 语法

```
1 SELECT 字段列表 FROM 表名 ORDER BY 字段1 排序方式1, 字段2 排序方式2 ;
```

2). 排序方式

- ASC：升序(默认值)
- DESC：降序

注意事项：

- 如果是升序，可以不指定排序方式ASC；
- 如果是多字段排序，当第一个字段值相同时，才会根据第二个字段进行排序；

案例：

A. 根据年龄对公司的员工进行升序排序

```
1 select * from emp order by age asc;  
2 select * from emp order by age;
```

B. 根据入职时间，对员工进行降序排序

```
1 select * from emp order by entrydate desc;
```

C. 根据年龄对公司的员工进行升序排序，年龄相同，再按照入职时间进行降序排序

```
1 select * from emp order by age asc , entrydate desc;
```

2.6.7 分页查询

分页操作在业务系统开发时，也是非常常见的一个功能，我们在网站中看到的各种各样的分页条，后台都需要借助于数据库的分页操作。



1). 语法

```
1 SELECT  字段列表 FROM  表名 LIMIT  起始索引, 查询记录数 ;
```

注意事项:

- 起始索引从0开始，起始索引 = (查询页码 - 1) * 每页显示记录数。
- 分页查询是数据库的方言，不同的数据库有不同的实现，MySQL中是LIMIT。
- 如果查询的是第一页数据，起始索引可以省略，直接简写为 limit 10。

案例:

A. 查询第1页员工数据，每页展示10条记录

```
1 select * from emp limit 0,10;
2 select * from emp limit 10;
```

B. 查询第2页员工数据，每页展示10条记录 -----> (页码-1) * 页展示记录数

```
1 select * from emp limit 10,10;
```

2.6.8 案例

1). 查询年龄为20,21,22,23岁的员工信息。

```
1 select * from emp where gender = '女' and age in(20,21,22,23);
```

2). 查询性别为 男 , 并且年龄在 20-40 岁(含)以内的姓名为三个字的员工。

```
1  select * from emp where gender = '男' and ( age between 20 and 40 ) and name like  
   '___';
```

3). 统计员工表中, 年龄小于60岁的 , 男性员工和女性员工的人数。

```
1  select gender, count(*) from emp where age < 60 group by gender;
```

4). 查询所有年龄小于等于35岁员工的姓名和年龄, 并对查询结果按年龄升序排序, 如果年龄相同按入职时间降序排序。

```
1  select name , age from emp where age <= 35 order by age asc , entrydate desc;
```

5). 查询性别为男, 且年龄在20-40 岁(含)以内的前5个员工信息, 对查询的结果按年龄升序排序, 年龄相同按入职时间升序排序。

```
1  select * from emp where gender = '男' and age between 20 and 40 order by age asc ,  
   entrydate asc limit 5 ;
```

2.6.9 执行顺序

在讲解DQL语句的具体语法之前, 我们已经讲解了DQL语句的完整语法, 及编写顺序, 接下来, 我们要说明的是DQL语句在执行时的执行顺序, 也就是先执行那一部分, 后执行那一部分。



验证:

查询年龄大于15的员工姓名、年龄, 并根据年龄进行升序排序。


```
1 select name , age from emp where age > 15 order by age asc;
```

在查询时，我们给emp表起一个别名 e，然后在select 及 where中使用该别名。

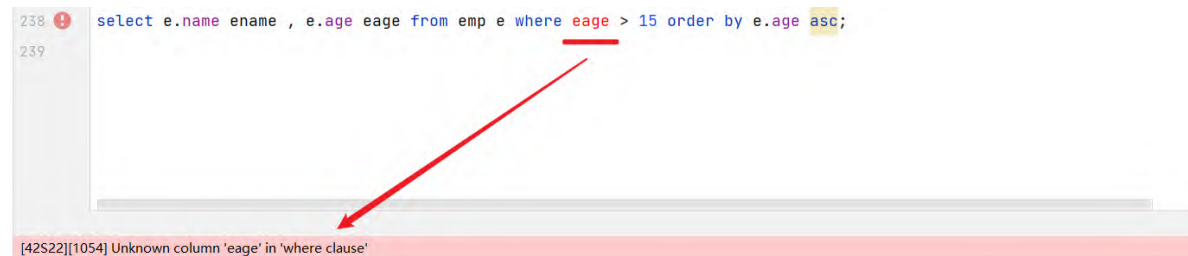
```
1 select e.name , e.age from emp e where e.age > 15 order by age asc;
```

执行上述SQL语句后，我们看到依然可以正常的查询到结果，此时就说明： from 先执行，然后 where 和 select 执行。那 where 和 select 到底哪个先执行呢？

此时，此时我们可以给select后面的字段起别名，然后在 where 中使用这个别名，然后看看是否可以执行成功。

```
1 select e.name ename , e.age eage from emp e where eage > 15 order by age asc;
```

执行上述SQL报错了：



由此我们可以得出结论： from 先执行，然后执行 where ， 再执行select 。

接下来，我们再执行如下SQL语句，查看执行效果：

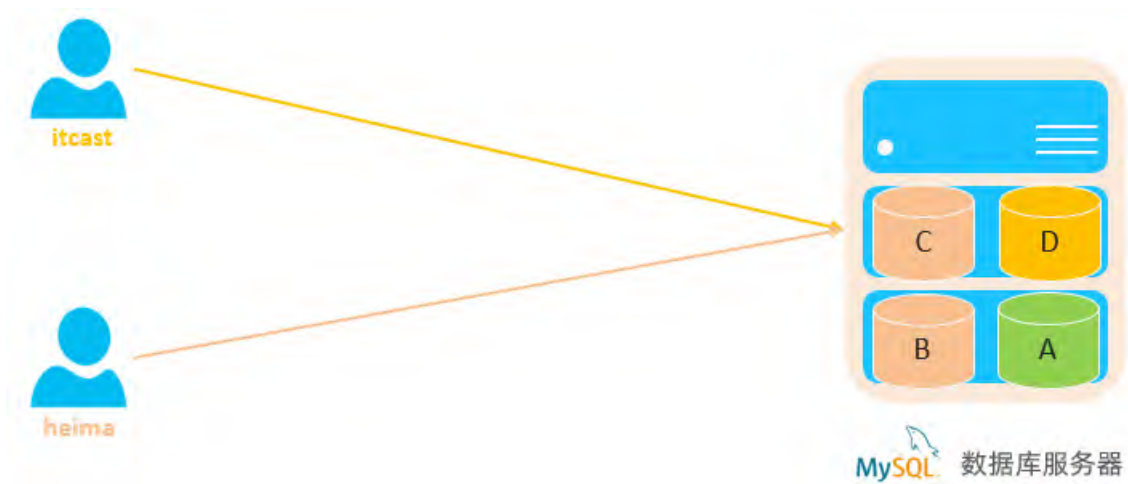
```
1 select e.name ename , e.age eage from emp e where e.age > 15 order by eage asc;
```

结果执行成功。 那么也就验证了： order by 是在select 语句之后执行的。

综上所述，我们可以看到DQL语句的执行顺序为： from ... where ... group by ...
having ... select ... order by ... limit ...

2.7 DCL

DCL英文全称是Data Control Language (数据控制语言)，用来管理数据库用户、控制数据库的访问权限。



2.7.1 管理用户

1). 查询用户

```
1 select * from mysql.user;
```

查询的结果如下:

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv
localhost	mysql.infoschema	Y	N	N	N	N	N
localhost	mysql.session	N	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y	Y

其中 Host代表当前用户访问的主机，如果为localhost，仅代表只能够在当前本机访问，是不可以远程访问的。 User代表的是访问该数据库的用户名。在MySQL中需要通过Host和用户来唯一标识一个用户。

2). 创建用户

```
1 CREATE USER '用户名'@'主机名' IDENTIFIED BY '密码';
```

3). 修改用户密码

```
1 ALTER USER '用户名'@'主机名' IDENTIFIED WITH mysql_native_password BY '新密码' ;
```

4). 删除用户

```
1 DROP USER '用户名'@'主机名' ;
```

注意事项:

- 在MySQL中需要通过用户名@主机名的方式，来唯一标识一个用户。

- 主机名可以使用 % 通配。
- 这类SQL开发人员操作的比较少，主要是DBA（Database Administrator 数据库管理员）使用。

案例：

A. 创建用户itcast，只能够在当前主机localhost访问，密码123456；

```
1 create user 'itcast'@'localhost' identified by '123456';
```

B. 创建用户heima，可以在任意主机访问该数据库，密码123456；

```
1 create user 'heima'@'%' identified by '123456';
```

C. 修改用户heima的访问密码为1234；

```
1 alter user 'heima'@'%' identified with mysql_native_password by '1234';
```

D. 删除 itcast@localhost 用户

```
1 drop user 'itcast'@'localhost';
```

2.7.2 权限控制

MySQL中定义了很多种权限，但是常用的就以下几种：

权限	说明
ALL, ALL PRIVILEGES	所有权限
SELECT	查询数据
INSERT	插入数据
UPDATE	修改数据
DELETE	删除数据
ALTER	修改表
DROP	删除数据库/表/视图
CREATE	创建数据库/表

上述只是简单罗列了常见的几种权限描述，其他权限描述及含义，可以直接参考 [官方文档](#)。

1). 查询权限

```
1 SHOW GRANTS FOR '用户名'@'主机名' ;
```

2). 授予权限

```
1 GRANT 权限列表 ON 数据库名.表名 TO '用户名'@'主机名';
```

3). 撤销权限

```
1 REVOKE 权限列表 ON 数据库名.表名 FROM '用户名'@'主机名';
```

注意事项：

- 多个权限之间，使用逗号分隔
- 授权时， 数据库名和表名可以使用 * 进行通配，代表所有。

案例：

A. 查询 'heima'@'%' 用户的权限

```
1 show grants for 'heima'@'%';
```

B. 授予 'heima'@'%' 用户itcast数据库所有表的所有操作权限

```
1 grant all on itcast.* to 'heima'@'%';
```

C. 撤销 'heima'@'%' 用户的itcast数据库的所有权限

```
1 revoke all on itcast.* from 'heima'@'%';
```

3. 函数

函数 是指一段可以直接被另一段程序调用的程序或代码。 也就意味着，这一段程序或代码在MySQL中已经给我们提供了，我们要做的就是合适的业务场景调用对应的函数完成对应的业务需求即可。 那么，函数到底在哪儿使用呢？

我们先来看两个场景：



1) . 在企业的OA或其他的人力系统中，经常会提供的有这样一个功能，每一个员工登录上来之后都能够看到当前员工入职的天数。 而在数据库中，存储的都是入职日期，如 2000-11-12，那如果快速计算出天数呢？

2) . 在做报表这类的业务需求中，我们要展示出学员的分数等级分布。而在数据库中，存储的是学生的分数值，如98/75，如何快速判定分数的等级呢？

其实，上述的这一类的需求呢，我们通过MySQL中的函数都可以很方便的实现 。

MySQL中的函数主要分为以下四类： 字符串函数、数值函数、日期函数、流程函数。

3.1 字符串函数

MySQL中内置了很多字符串函数，常用的几个如下：

函数	功能
CONCAT(S1,S2,...Sn)	字符串拼接，将S1, S2, ... Sn拼接成一个字符串
LOWER(str)	将字符串str全部转为小写
UPPER(str)	将字符串str全部转为大写
LPAD(str,n,pad)	左填充，用字符串pad对str的左边进行填充，达到n个字符串长度
RPAD(str,n,pad)	右填充，用字符串pad对str的右边进行填充，达到n个字符串长度
TRIM(str)	去掉字符串头部和尾部的空格
SUBSTRING(str,start,len)	返回从字符串str从start位置起的len个长度的字符串

演示如下：

A. concat ： 字符串拼接

```
1 select concat('Hello' , ' MySQL');
```

B. lower ： 全部转小写

```
1 select lower('Hello');
```

C. upper ： 全部转大写

```
1 select upper('Hello');
```

D. lpad ： 左填充

```
1 select lpad('01', 5, '-');
```

E. rpad ： 右填充

```
1 select rpad('01', 5, '-');
```

F. trim ： 去除空格

```
1 select trim(' Hello MySQL ');
```

G. substring ： 截取子字符串

```
1 select substring('Hello MySQL',1,5);
```

案例：

id	workno	name	gender	age	idcard	workaddress	entrydate
1	1	柳岩	女	20	123456789012345678	北京	2000-01-01
2	2	张无忌	男	18	123456789012345670	北京	2005-09-01
3	3	韦一笑	男	38	123456789712345670	上海	2005-08-01
4	4	赵敏	女	18	123456757123845670	北京	2009-12-01
5	5	小昭	女	16	123456769012345678	上海	2007-07-01
6	6	杨逍	男	28	12345678931234567X	北京	2006-01-01
7	7	范瑶	男	40	123456789212345670	北京	2005-05-01
8	8	黛绮丝	女	38	123456157123645670	天津	2015-05-01
9	9	范凉凉	女	45	123156789012345678	北京	2010-04-01
10	10	陈友谅	男	53	123456789012345670	上海	2011-01-01
11	11	张士诚	男	55	123567897123465670	江苏	2015-05-01
12	12	常遇春	男	32	123446757152345670	北京	2004-02-01
13	13	张三丰	男	88	123656789012345678	江苏	2020-11-01
14	14	灭绝	女	65	123456719012345670	西安	2019-05-01
15	15	胡青牛	男	70	12345674971234567X	西安	2018-04-01
16	16	周芷若	女	18	<null>	北京	2012-06-01

由于业务需求变更，企业员工的工号，统一为5位数，目前不足5位数的全部在前面补0。比如：1号员工的工号应该为00001。

```
1 update emp set workno = lpad(workno, 5, '0');
```

处理完毕后，具体的数据为：

id	workno	name	gender	age	idcard	workaddress	entrydate
1	00001	柳岩	女	20	123456789012345678	北京	2000-01-01
2	00002	张无忌	男	18	123456789012345670	北京	2005-09-01
3	00003	韦一笑	男	38	123456789712345670	上海	2005-08-01
4	00004	赵敏	女	18	123456757123845670	北京	2009-12-01
5	00005	小昭	女	16	123456769012345678	上海	2007-07-01
6	00006	杨逍	男	28	12345678931234567X	北京	2006-01-01
7	00007	范瑶	男	40	123456789212345670	北京	2005-05-01
8	00008	黛绮丝	女	38	123456157123645670	天津	2015-05-01
9	00009	范凉凉	女	45	123156789012345678	北京	2010-04-01
10	00010	陈友谅	男	53	123456789012345670	上海	2011-01-01
11	00011	张士诚	男	55	123567897123465670	江苏	2015-05-01
12	00012	常遇春	男	32	123446757152345670	北京	2004-02-01
13	00013	张三丰	男	88	123656789012345678	江苏	2020-11-01
14	00014	灭绝	女	65	123456719012345670	西安	2019-05-01
15	00015	胡青牛	男	70	12345674971234567X	西安	2018-04-01
16	00016	周芷若	女	18	<null>	北京	2012-06-01

3.2 数值函数

常见的数值函数如下：

函数	功能
CEIL(x)	向上取整
FLOOR(x)	向下取整
MOD(x,y)	返回x/y的模
RAND()	返回0~1内的随机数
ROUND(x,y)	求参数x的四舍五入的值，保留y位小数

演示如下：

A. ceil：向上取整

```
1 select ceil(1.1);
```

B. floor：向下取整

```
1 select floor(1.9);
```

C. mod：取模

```
1 select mod(7,4);
```

D. rand：获取随机数

```
1 select rand();
```

E. round：四舍五入

```
1 select round(2.344,2);
```

案例：

通过数据库的函数，生成一个六位数的随机验证码。

思路： 获取随机数可以通过rand() 函数，但是获取出来的随机数是在0-1之间的，所以可以在其基础上乘以1000000，然后舍弃小数部分，如果长度不足6位，补0

```
1 select lpad(round(rand()*1000000 , 0) , 6, '0');
```

3.3 日期函数

常见的日期函数如下：

函数	功能
CURDATE()	返回当前日期
CURTIME()	返回当前时间
NOW()	返回当前日期和时间
YEAR(date)	获取指定date的年份
MONTH(date)	获取指定date的月份
DAY(date)	获取指定date的日期
DATE_ADD(date, INTERVAL expr type)	返回一个日期/时间值加上一个时间间隔expr后的时间值
DATEDIFF(date1,date2)	返回起始时间date1 和 结束时间date2之间的天数

演示如下：

A. curdate: 当前日期

```
1 select curdate();
```

B. curtime: 当前时间

```
1 select curtime();
```

C. now: 当前日期和时间

```
1 select now();
```

D. YEAR , MONTH , DAY: 当前年、月、日

```
1 select YEAR(now());
2 select MONTH(now());
3 select DAY(now());
```

E. date_add: 增加指定的时间间隔

```
1 select date_add(now(), INTERVAL 70 YEAR );
```

F. datediff: 获取两个日期相差的天数

```
1 select datediff('2021-10-01', '2021-12-01');
```

案例:

查询所有员工的入职天数, 并根据入职天数倒序排序。

思路: 入职天数, 就是通过当前日期 - 入职日期, 所以需要使用datediff函数来完成。

```
1 select name, datediff(curdate(), entrydate) as 'entrydays' from emp order by  
entrydays desc;
```

3.4 流程函数

流程函数也是很常用的一类函数, 可以在SQL语句中实现条件筛选, 从而提高语句的效率。

函数	功能
IF(value , t , f)	如果value为true, 则返回t, 否则返回f
IFNULL(value1 , value2)	如果value1不为空, 返回value1, 否则返回value2
CASE WHEN [val1] THEN [res1] ... ELSE [default] END	如果val1为true, 返回res1, ... 否则返回default默认值
CASE [expr] WHEN [val1] THEN [res1] ... ELSE [default] END	如果expr的值等于val1, 返回res1, ... 否则返回default默认值

演示如下:

A. if

```
1 select if(false, 'Ok', 'Error');
```

B. ifnull

```
1  select ifnull('Ok','Default');
2
3  select ifnull('','Default');
4
5  select ifnull(null,'Default');
```

C. case when then else end

需求：查询emp表的员工姓名和工作地址（北京/上海 ----> 一线城市，其他 ----> 二线城市）

```
1  select
2      name,
3      ( case workaddress when '北京' then '一线城市' when '上海' then '一线城市' else
4          '二线城市' end ) as '工作地址'
5  from emp;
```

案例：

```
1  create table score(
2      id int comment 'ID',
3      name varchar(20) comment '姓名',
4      math int comment '数学',
5      english int comment '英语',
6      chinese int comment '语文'
7  ) comment '学员成绩表';
8  insert into score(id, name, math, english, chinese) VALUES (1, 'Tom', 67, 88, 95
9      ), (2, 'Rose' , 23, 66, 90), (3, 'Jack', 56, 98, 76);
```

具体的SQL语句如下：

```
1  select
2      id,
3      name,
4      (case when math >= 85 then '优秀' when math >=60 then '及格' else '不及格' end )
      '数学',
5      (case when english >= 85 then '优秀' when english >=60 then '及格' else '不及格'
      end ) '英语',
6      (case when chinese >= 85 then '优秀' when chinese >=60 then '及格' else '不及格'
      end ) '语文'
7  from score;
```

MySQL的常见函数我们学习完了，那接下来，我们就来分析一下，在前面讲到的两个函数的案例场景，思考一下需要用到什么样的函数来实现？

1). 数据库中，存储的是入职日期，如 2000-01-01，如何快速计算出入职天数呢？ ----->

答案：datediff

2). 数据库中，存储的是学生的分数值，如98、75，如何快速判定分数的等级呢？ ----->

答案：case ... when ...

4. 约束

4.1 概述

概念：约束是作用于表中字段上的规则，用于限制存储在表中的数据。

目的：保证数据库中数据的正确、有效性和完整性。

分类：

约束	描述	关键字
非空约束	限制该字段的数据不能为null	NOT NULL
唯一约束	保证该字段的所有数据都是唯一、不重复的	UNIQUE
主键约束	主键是一行数据的唯一标识，要求非空且唯一	PRIMARY KEY
默认约束	保存数据时，如果未指定该字段的值，则采用默认值	DEFAULT
检查约束 (8.0.16版本之后)	保证字段值满足某一个条件	CHECK
外键约束	用来让两张表的数据之间建立连接，保证数据的一致性和完整性	FOREIGN KEY

注意：约束是作用于表中字段上的，可以在创建表/修改表的时候添加约束。

4.2 约束演示

上面我们介绍了数据库中常见的约束，以及约束涉及到的关键字，那这些约束我们到底如何在创建表、修改表的时候来指定呢，接下来我们就通过一个案例，来演示一下。

案例需求： 根据需求，完成表结构的创建。需求如下：

字段名	字段含义	字段类型	约束条件	约束关键字
id	ID唯一标识	int	主键，并且自动增长	PRIMARY KEY, AUTO_INCREMENT
name	姓名	varchar(10)	不为空，并且唯一	NOT NULL , UNIQUE
age	年龄	int	大于0，并且小于等于120	CHECK
status	状态	char(1)	如果没有指定该值，默认为1	DEFAULT
gender	性别	char(1)	无	

对应的建表语句为：

```

1  CREATE TABLE tb_user(
2      id int AUTO_INCREMENT PRIMARY KEY COMMENT 'ID唯一标识',
3      name varchar(10) NOT NULL UNIQUE COMMENT '姓名' ,
4      age int check (age > 0 && age <= 120) COMMENT '年龄' ,
5      status char(1) default '1' COMMENT '状态',
6      gender char(1) COMMENT '性别'
7  );

```

在为字段添加约束时，我们只需要在字段之后加上约束的关键字即可，需要关注其语法。我们执行上面的SQL把表结构创建完成，然后接下来，就可以通过一组数据进行测试，从而验证一下，约束是否可以生效。

```

1  insert into tb_user(name,age,status,gender) values ('Tom1',19,'1','男'),
    ('Tom2',25,'0','男');
2  insert into tb_user(name,age,status,gender) values ('Tom3',19,'1','男');
3
4  insert into tb_user(name,age,status,gender) values (null,19,'1','男');
5  insert into tb_user(name,age,status,gender) values ('Tom3',19,'1','男');
6
7  insert into tb_user(name,age,status,gender) values ('Tom4',80,'1','男');
8  insert into tb_user(name,age,status,gender) values ('Tom5',-1,'1','男');
9  insert into tb_user(name,age,status,gender) values ('Tom5',121,'1','男');
10
11 insert into tb_user(name,age,gender) values ('Tom5',120,'男');

```

上面，我们是通过编写SQL语句的形式来完成约束的指定，那假如我们是通过图形化界面来创建表结构时，又该如何来指定约束呢？只需要在创建表的时候，根据我们的需要选择对应的约束即可。

准备数据

```
1  create table dept(
2      id    int auto_increment comment 'ID' primary key,
3      name  varchar(50) not null comment '部门名称'
4  )comment '部门表';
5  INSERT INTO dept (id, name) VALUES (1, '研发部'), (2, '市场部'), (3, '财务部'), (4,
6      '销售部'), (5, '总经办');
7
8  create table emp(
9      id    int auto_increment comment 'ID' primary key,
10     name  varchar(50) not null comment '姓名',
11     age   int comment '年龄',
12     job   varchar(20) comment '职位',
13     salary int comment '薪资',
14     entrydate date comment '入职时间',
15     managerid int comment '直属领导ID',
16     dept_id int comment '部门ID'
17 )comment '员工表';
18
19 INSERT INTO emp (id, name, age, job, salary, entrydate, managerid, dept_id)
20 VALUES
21     (1, '金庸', 66, '总裁', 20000, '2000-01-01', null, 5), (2, '张无忌', 20,
22     '项目经理', 12500, '2005-12-05', 1, 1),
23     (3, '杨逍', 33, '开发', 8400, '2000-11-03', 2, 1), (4, '韦一笑', 48, '开
24     发', 11000, '2002-02-05', 2, 1),
25     (5, '常遇春', 43, '开发', 10500, '2004-09-07', 3, 1), (6, '小昭', 19, '程
26     序员鼓励师', 6600, '2004-10-12', 2, 1);
```

emp 员工表

id	name	age	job	salary	entrydate	managerid	dept_id
1	金庸	66	总裁	20000	2000-01-01	<null>	5
2	张无忌	20	项目经理	12500	2005-12-05	1	1
3	杨逍	33	开发	8400	2000-11-03	2	1
4	韦一笑	48	开发	11000	2002-02-05	2	1
5	常遇春	43	开发	10500	2004-09-07	3	1
6	小昭	19	程序员鼓励师	6600	2004-10-12	2	1

dept 部门表

id	name
1	研发部
2	市场部
3	财务部
4	销售部
5	总经办

接下来，我们可以做一个测试，删除id为1的部门信息。

emp 员工表

id	name	age	job	salary	entrydate	managerid	dept_id
1	金庸	66	总裁	20000	2000-01-01	<null>	5
2	张无忌	20	项目经理	12500	2005-12-05	1	1
3	杨逍	33	开发	8400	2000-11-03	2	1
4	韦一笑	48	开发	11000	2002-02-05	2	1
5	常遇春	43	开发	10500	2004-09-07	3	1
6	小昭	19	程序员鼓励师	6600	2004-10-12	2	1

dept 部门表

id	name
2	市场部
3	财务部
4	销售部
5	总经办

结果，我们看到删除成功，而删除成功之后，部门表不存在id为1的部门，而在emp表中还有很多的员工，关联的为id为1的部门，此时就出现了数据的不完整性。 而要想解决这个问题就得通过数据库的外键约束。

4.3.2 语法

1). 添加外键

```

1 CREATE TABLE 表名 (
2     字段名      数据类型,
3     ...
4     [CONSTRAINT]  [外键名称] FOREIGN KEY (外键字段名) REFERENCES 主表 (主表列名)
5 );

```

```

1 ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段名)
REFERENCES 主表 (主表列名) ;

```

案例：

为emp表的dept_id字段添加外键约束,关联dept表的主键id。

```

1 alter table emp add constraint fk_emp_dept_id foreign key (dept_id) references
dept(id);

```

id	name	age	job	salary	entrydate	managerid	dept_id
1	金庸	66	总裁	20000	2000-01-01	<null>	5
2	张无忌	20	项目经理	12500	2005-12-05	1	1
3	杨逍	33	开发	8400	2000-11-03	2	1
4	韦一笑	48	开发	11000	2002-02-05	2	1
5	常遇春	43	开发	10500	2004-09-07	3	1
6	小昭	19	程序员鼓励师	6600	2004-10-12	2	1

添加了外键约束之后，我们再到dept表(父表)删除id为1的记录，然后看一下会发生什么现象。 此时将会报错，不能删除或更新父表记录，因为存在外键约束。

[23000][1451] Cannot delete or update a parent row: a foreign key constraint fails ('itcast'.emp, CONSTRAINT 'fk_emp_dept_id' FOREIGN KEY ('dept_id') REFERENCES 'dept' ('id'))

2). 删除外键

```
1 ALTER TABLE 表名 DROP FOREIGN KEY 外键名称;
```

案例:

删除emp表的外键fk_emp_dept_id。

```
1 alter table emp drop foreign key fk_emp_dept_id;
```

4.3.3 删除/更新行为

添加了外键之后，再删除父表数据时产生的约束行为，我们就称为删除/更新行为。具体的删除/更新行为有以下几种：

行为	说明
NO ACTION	当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则不允许删除/更新。（与 RESTRICT 一致）默认行为
RESTRICT	当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则不允许删除/更新。（与 NO ACTION 一致）默认行为
CASCADE	当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有，则也删除/更新外键在子表中的记录。
SET NULL	当在父表中删除对应记录时，首先检查该记录是否有对应外键，如果有则设置子表中该外键值为null（这就要求该外键允许取null）。
SET DEFAULT	父表有变更时，子表将外键列设置成一个默认的值（Innodb不支持）

具体语法为：

```
1 ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段) REFERENCES  
主表名 (主表字段名) ON UPDATE CASCADE ON DELETE CASCADE;
```

演示如下：

由于NO ACTION 是默认行为，我们前面语法演示的时候，已经测试过了，就不再演示了，这里我们再演示其他的两种行为：CASCADE、SET NULL。

1) . CASCADE

```
1 alter table emp add constraint fk_emp_dept_id foreign key (dept_id) references  
dept(id) on update cascade on delete cascade ;
```

A. 修改父表id为1的记录，将id修改为6

id	name	age	job	salary	entrydate	managerid	dept_id
2	市场部						5
3	财务部						6
4	销售部						6
5	总经办						6
6	研发部						6

id	name	age	job	salary	entrydate	managerid	dept_id
1	金庸	66	总裁	20000	2000-01-01	<null>	5
2	张无忌	20	项目经理	12500	2005-12-05	1	6
3	杨逍	33	开发	8400	2000-11-03	2	6
4	韦一笑	48	开发	11000	2002-02-05	2	6
5	常遇春	43	开发	10500	2004-09-07	3	6
6	小昭	19	程序员鼓励师	6600	2004-10-12	2	6

我们发现，原来在子表中dept_id值为1的记录，现在也变为6了，这就是cascade级联的效果。

在一般的业务系统中，不会修改一张表的主键值。

B. 删除父表id为6的记录

id	name
2	市场部
3	财务部
4	销售部
5	总经办

id	name	age	job	salary	entrydate	managerid	dept_id
1	金庸	66	总裁	20000	2000-01-01	<null>	5

我们发现，父表的数据删除成功了，但是子表中关联的记录也被级联删除了。

2) . SET NULL

在进行测试之前，我们先需要删除上面建立的外键 fk_emp_dept_id。然后再通过数据脚本，将 emp、dept表的数据恢复了。

```
1 alter table emp add constraint fk_emp_dept_id foreign key (dept_id) references  
dept(id) on update set null on delete set null ;
```

接下来，我们删除id为1的数据，看看会发生什么样的现象。

id	name
1	研发部
2	市场部
3	财务部
4	销售部
5	总经办

→

id	name
2	市场部
3	财务部
4	销售部
5	总经办

我们发现父表的记录是可以正常的删除的，父表的数据删除之后，再打开子表 emp，我们发现子表 emp 的 dept_id 字段，原来 dept_id 为 1 的数据，现在都被置为 NULL 了。

id	name	age	job	salary	entrydate	managerid	dept_id
1	金庸	66	总裁	20000	2000-01-01	<null>	5
2	张无忌	20	项目经理	12500	2005-12-05	1	<null>
3	杨逍	33	开发	8400	2000-11-03	2	<null>
4	韦一笑	48	开发	11000	2002-02-05	2	<null>
5	常遇春	43	开发	10500	2004-09-07	3	<null>
6	小昭	19	程序员鼓励师	6600	2004-10-12	2	<null>

这就是 SET NULL 这种删除/更新行为的效果。

5. 多表查询

我们之前在讲解 SQL 语句的时候，讲解了 DQL 语句，也就是数据查询语句，但是之前讲解的查询都是单表查询，而本章节我们要学习的则是多表查询操作，主要从以下几个方面进行讲解。

5.1 多表关系

项目开发中，在进行数据库表结构设计时，会根据业务需求及业务模块之间的关系，分析并设计表结构，由于业务之间相互关联，所以各个表结构之间也存在着各种联系，基本上分为三种：

- 一对多 (多对一)
- 多对多
- 一对一

5.1.1 一对多

- 案例：部门 与 员工的关系
- 关系：一个部门对应多个员工，一个员工对应一个部门
- 实现：在多的-一方建立外键，指向一的一方的主键

员工表(emp) N				1 部门表(dept)	
id	name	age	dept_id	id	name
1	张无忌	20	1	1	研发部
2	杨逍	33	1	2	市场部
3	赵敏	18	2	3	财务部
4	常遇春	43	2	4	销售部

5.1.2 多对多

- 案例：学生 与 课程的关系
- 关系：一个学生可以选修多门课程，一门课程也可以供多个学生选择
- 实现：建立第三张中间表，中间表至少包含两个外键，分别关联两方主键

学生表(student) N			学生课程关系表(student_course)			N 课程表(course)	
id	name	no	studentid	courseid		id	name
1	黛绮丝	2000100101	1	1		1	Java
2	谢逊	2000100102	1	2		2	PHP
3	殷天正	2000100103	1	3		3	MySQL
4	韦一笑	2000100104	2	1		4	Hadoop
			2	4			

对应的SQL脚本：

```

1  create table student(
2      id int auto_increment primary key comment '主键ID',
3      name varchar(10) comment '姓名',
4      no varchar(10) comment '学号'
5  ) comment '学生表';
6  insert into student values (null, '黛绮丝', '2000100101'), (null, '谢逊',
7      '2000100102'), (null, '殷天正', '2000100103'), (null, '韦一笑', '2000100104');
8
9  create table course(
10     id int auto_increment primary key comment '主键ID',
11     name varchar(10) comment '课程名称'
12 ) comment '课程表';
13 insert into course values (null, 'Java'), (null, 'PHP'), (null, 'MySQL'),
14     (null, 'Hadoop');
15
16 create table student_course(

```



```

17         id int auto_increment comment '主键' primary key,
18         studentid int not null comment '学生ID',
19         courseid int not null comment '课程ID',
20         constraint fk_courseid foreign key (courseid) references course (id),
21         constraint fk_studentid foreign key (studentid) references student (id)
22     ) comment '学生课程中间表';
23
24     insert into student_course values (null,1,1), (null,1,2), (null,1,3), (null,2,2),
        (null,2,3), (null,3,4);

```

5.1.3 一对一

- 案例：用户 与 用户详情的关系
- 关系：一对一关系，多用于单表拆分，将一张表的基础字段放在一张表中，其他详情字段放在另一张表中，以提升操作效率
- 实现：在任意一方加入外键，关联另外一方的主键，并且设置外键为唯一的 (UNIQUE)

id	name	age	gender	phone
1	黄渤	45	1	18800001111
2	冰冰	35	2	18800002222
3	马云	55	1	18800008888
4	李彦宏	50	1	18800009999

用户基本信息表(tb_user)

id	degree	major	primaryschool	middleschool	university	userid
1	本科	舞蹈	静安区第一小学	静安区第一中学	北京舞蹈学院	1
2	硕士	表演	朝阳区第一小学	朝阳区第一中学	北京电影学院	2
3	本科	英语	杭州市第一小学	杭州市第一中学	杭州师范大学	3
4	本科	应用数学	阳泉第一小学	阳泉区第一中学	清华大学	4

用户教育信息表(tb_user_edu)

对应的SQL脚本：

```

1     create table tb_user(
2         id int auto_increment primary key comment '主键ID',
3         name varchar(10) comment '姓名',
4         age int comment '年龄',
5         gender char(1) comment '1: 男 , 2: 女',
6         phone char(11) comment '手机号'
7     ) comment '用户基本信息表';
8
9     create table tb_user_edu(
10        id int auto_increment primary key comment '主键ID',
11        degree varchar(20) comment '学历',
12        major varchar(50) comment '专业',
13        primaryschool varchar(50) comment '小学',
14        middleschool varchar(50) comment '中学',

```

```

15     university varchar(50) comment '大学',
16     userid int unique comment '用户ID',
17     constraint fk_userid foreign key (userid) references tb_user(id)
18 ) comment '用户教育信息表';
19
20
21 insert into tb_user(id, name, age, gender, phone) values
22     (null, '黄渤', 45, '1', '18800001111'),
23     (null, '冰冰', 35, '2', '18800002222'),
24     (null, '码云', 55, '1', '18800008888'),
25     (null, '李彦宏', 50, '1', '18800009999');
26
27 insert into tb_user_edu(id, degree, major, primaryschool, middleschool,
28     university, userid) values
29     (null, '本科', '舞蹈', '静安区第一小学', '静安区第一中学', '北京舞蹈学院', 1),
30     (null, '硕士', '表演', '朝阳区第一小学', '朝阳区第一中学', '北京电影学院', 2),
31     (null, '本科', '英语', '杭州市第一小学', '杭州市第一中学', '杭州师范大学', 3),
32     (null, '本科', '应用数学', '阳泉第一小学', '阳泉区第一中学', '清华大学', 4);

```

5.2 多表查询概述

5.2.1 数据准备

- 1). 删除之前 emp, dept表的测试数据
- 2). 执行如下脚本, 创建emp表与dept表并插入测试数据

```

1  -- 创建dept表, 并插入数据
2  create table dept(
3      id    int auto_increment comment 'ID' primary key,
4      name  varchar(50) not null comment '部门名称'
5  ) comment '部门表';
6  INSERT INTO dept (id, name) VALUES (1, '研发部'), (2, '市场部'), (3, '财务部'), (4,
7      '销售部'), (5, '总经办'), (6, '人事部');
8
9  -- 创建emp表, 并插入数据
10 create table emp(
11     id    int auto_increment comment 'ID' primary key,

```

```

11     name varchar(50) not null comment '姓名',
12     age int comment '年龄',
13     job varchar(20) comment '职位',
14     salary int comment '薪资',
15     entrydate date comment '入职时间',
16     managerid int comment '直属领导ID',
17     dept_id int comment '部门ID'
18 )comment '员工表';
19 -- 添加外键
20 alter table emp add constraint fk_emp_dept_id foreign key (dept_id) references
    dept(id);
21
22 INSERT INTO emp (id, name, age, job,salary, entrydate, managerid, dept_id)
    VALUES
23     (1, '金庸', 66, '总裁',20000, '2000-01-01', null,5),
24     (2, '张无忌', 20, '项目经理',12500, '2005-12-05', 1,1),
25     (3, '杨逍', 33, '开发', 8400,'2000-11-03', 2,1),
26     (4, '韦一笑', 48, '开发',11000, '2002-02-05', 2,1),
27     (5, '常遇春', 43, '开发',10500, '2004-09-07', 3,1),
28     (6, '小昭', 19, '程序员鼓励师',6600, '2004-10-12', 2,1),
29     (7, '灭绝', 60, '财务总监',8500, '2002-09-12', 1,3),
30     (8, '周芷若', 19, '会计',48000, '2006-06-02', 7,3),
31     (9, '丁敏君', 23, '出纳',5250, '2009-05-13', 7,3),
32     (10, '赵敏', 20, '市场部总监',12500, '2004-10-12', 1,2),
33     (11, '鹿杖客', 56, '职员',3750, '2006-10-03', 10,2),
34     (12, '鹤笔翁', 19, '职员',3750, '2007-05-09', 10,2),
35     (13, '方东白', 19, '职员',5500, '2009-02-12', 10,2),
36     (14, '张三丰', 88, '销售总监',14000, '2004-10-12', 1,4),
37     (15, '俞莲舟', 38, '销售',4600, '2004-10-12', 14,4),
38     (16, '宋远桥', 40, '销售',4600, '2004-10-12', 14,4),
39     (17, '陈友谅', 42, null,2000, '2011-10-12', 1,null);

```

dept表共6条记录，emp表共17条记录。

5.2.2 概述

多表查询就是指从多张表中查询数据。

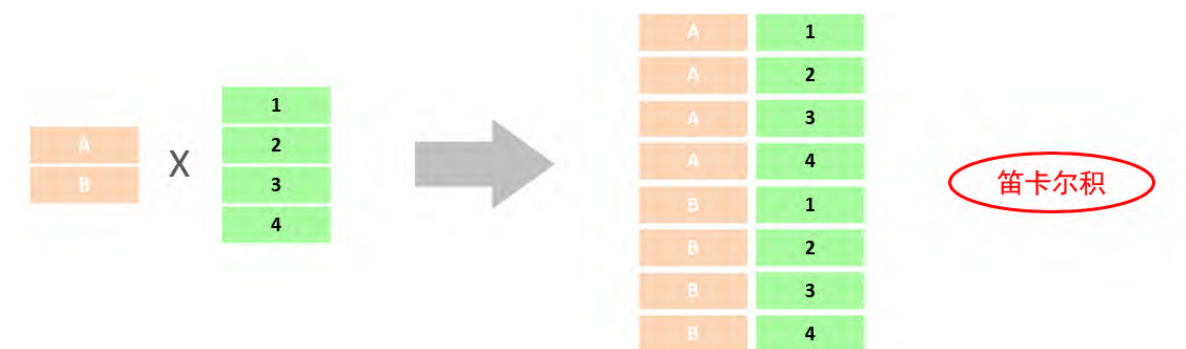
原来查询单表数据，执行的SQL形式为：select * from emp;

那么我们要执行多表查询，就只需要使用逗号分隔多张表即可，如：select * from emp , dept ; 具体的执行结果如下：

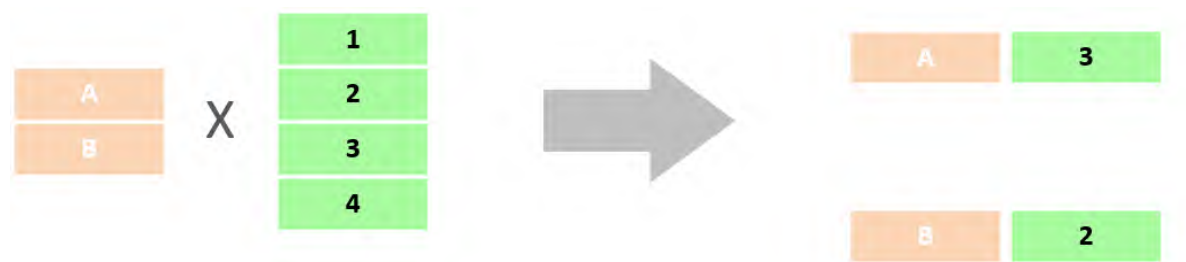
emp.id	emp.name	age	job	salary	entrydate	managerid	dept_id	dept.id	dept.name
1	金庸	66	总裁	20000	2000-01-01	<null>	5	6	人事部
1	金庸	66	总裁	20000	2000-01-01	<null>	5	5	总经办
1	金庸	66	总裁	20000	2000-01-01	<null>	5	4	销售部
1	金庸	66	总裁	20000	2000-01-01	<null>	5	3	财务部
1	金庸	66	总裁	20000	2000-01-01	<null>	5	2	市场部
1	金庸	66	总裁	20000	2000-01-01	<null>	5	1	研发部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	6	人事部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	5	总经办
2	张无忌	20	项目经理	12500	2005-12-05	1	1	4	销售部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	3	财务部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	2	市场部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	1	研发部
3	杨逍	33	开发	8400	2000-11-03	2	1	6	人事部
3	杨逍	33	开发	8400	2000-11-03	2	1	5	总经办

此时,我们看到查询结果中包含了大量的结果集，总共102条记录，而这其实就是员工表emp所有的记录（17）与 部门表dept所有记录（6）的所有组合情况，这种现象称之为笛卡尔积。接下来，就来简单介绍下笛卡尔积。

笛卡尔积：笛卡尔乘积是指在数学中，两个集合A集合 和 B集合的所有组合情况。



而在多表查询中，我们是需要消除无效的笛卡尔积的，只保留两张表关联部分的数据。



emp.id	emp.name	age	job	salary	entrydate	managerid	dept_id	dept.id	dept.name
1	金庸	66	总裁	20000	2000-01-01	<null>	5	6	人事部
1	金庸	66	总裁	20000	2000-01-01	<null>	5	5	总经办
1	金庸	66	总裁	20000	2000-01-01	<null>	5	4	销售部
1	金庸	66	总裁	20000	2000-01-01	<null>	5	3	财务部
1	金庸	66	总裁	20000	2000-01-01	<null>	5	2	市场部
1	金庸	66	总裁	20000	2000-01-01	<null>	5	1	研发部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	6	人事部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	5	总经办
2	张无忌	20	项目经理	12500	2005-12-05	1	1	4	销售部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	3	财务部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	2	市场部
2	张无忌	20	项目经理	12500	2005-12-05	1	1	1	研发部
3	杨逍	33	开发	8400	2000-11-03	2	1	6	人事部
3	杨逍	33	开发	8400	2000-11-03	2	1	5	总经办

在SQL语句中，如何去去除无效的笛卡尔积呢？ 我们可以给多表查询加上连接查询的条件即可。

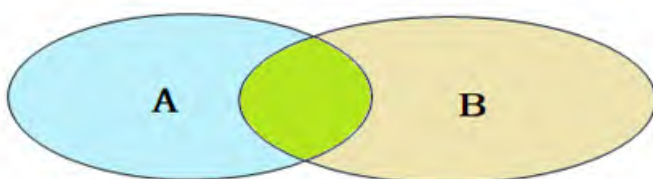
```
select * from emp , dept where emp.dept_id = dept.id;
```

emp.id	emp.name	age	job	salary	entrydate	managerid	dept_id	dept.id	dept.name
1	金庸	66	总裁	20000	2000-01-01	<null>	5	5	总经办
2	张无忌	20	项目经理	12500	2005-12-05	1	1	1	研发部
3	杨逍	33	开发	8400	2000-11-03	2	1	1	研发部
4	韦一笑	48	开发	11000	2002-02-05	2	1	1	研发部
5	常遇春	43	开发	10500	2004-09-07	3	1	1	研发部
6	小昭	19	程序员鼓励师	6600	2004-10-12	2	1	1	研发部
7	灭绝	60	财务总监	8500	2002-09-12	1	3	3	财务部
8	周芷若	19	会计	48000	2006-06-02	7	3	3	财务部
9	丁敏君	23	出纳	5250	2009-05-13	7	3	3	财务部
10	赵敏	20	市场部总监	12500	2004-10-12	1	2	2	市场部
11	鹿杖客	56	职员	3750	2006-10-03	10	2	2	市场部
12	鹤笔翁	19	职员	3750	2007-05-09	10	2	2	市场部
13	方东白	19	职员	5500	2009-02-12	10	2	2	市场部
14	张三丰	88	销售总监	14000	2004-10-12	1	4	4	销售部
15	俞莲舟	38	销售	4600	2004-10-12	14	4	4	销售部
16	宋远桥	40	销售	4600	2004-10-12	14	4	4	销售部

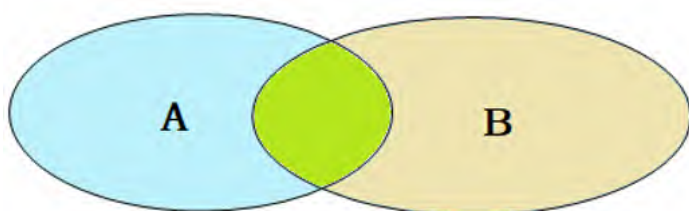
而由于id为17的员工，没有dept_id字段值，所以在多表查询时，根据连接查询的条件并没有查询到。

5.2.3 分类

- 连接查询
 - 内连接：相当于查询A、B交集部分数据
 - 外连接：
 - 左外连接：查询左表所有数据，以及两张表交集部分数据
 - 右外连接：查询右表所有数据，以及两张表交集部分数据
 - 自连接：当前表与自身的连接查询，自连接必须使用表别名
- 子查询



5.3 内连接



内连接查询的是两张表交集部分的数据。

据。（也就是绿色部分的数据）

内连接的语法分为两种：隐式内连接、显式内连接。先来学习一下具体的语法结构。

1). 隐式内连接

```
1  SELECT  字段列表  FROM  表1 , 表2  WHERE  条件 ... ;
```

2). 显式内连接

```
1  SELECT  字段列表  FROM  表1  [ INNER ] JOIN 表2  ON  连接条件 ... ;
```

案例:

A. 查询每一个员工的姓名 , 及关联的部门的名称 (隐式内连接实现)

表结构: emp , dept

连接条件: emp.dept_id = dept.id

```
1  select emp.name , dept.name from emp , dept where emp.dept_id = dept.id ;
2
3  -- 为每一张表起别名,简化SQL编写
4  select e.name,d.name from emp e , dept d where e.dept_id = d.id;
```

B. 查询每一个员工的姓名 , 及关联的部门的名称 (显式内连接实现) --- INNER JOIN ... ON ...

表结构: emp , dept

连接条件: emp.dept_id = dept.id

```
1  select e.name, d.name from emp e inner join dept d on e.dept_id = d.id;
2
3  -- 为每一张表起别名,简化SQL编写
4  select e.name, d.name from emp e join dept d on e.dept_id = d.id;
```

表的别名:

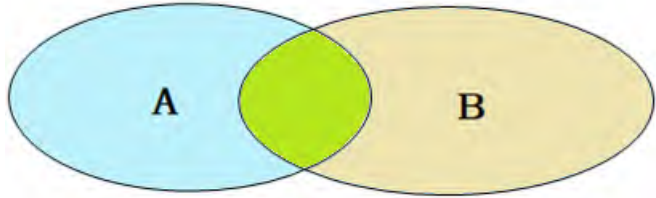
①. tablea as 别名1 , tableb as 别名2 ;

②. tablea 别名1 , tableb 别名2 ;

注意事项:

一旦为表起了别名，就不能再使用表名来指定对应的字段了，此时只能使用别名来指定字段。

5.4 外连接



外连接分为两种，分别是：左外连接 和 右外连接。具体的语法结构为：

1) . 左外连接

```
1  SELECT  字段列表  FROM  表1  LEFT  [ OUTER ]  JOIN  表2  ON  条件 ... ;
```

左外连接相当于查询表1 (左表) 的所有数据，当然也包含表1和表2交集部分的数据。

2) . 右外连接

```
1  SELECT  字段列表  FROM  表1  RIGHT  [ OUTER ]  JOIN  表2  ON  条件 ... ;
```

右外连接相当于查询表2 (右表) 的所有数据，当然也包含表1和表2交集部分的数据。

案例：

A. 查询emp表的所有数据，和对应的部门信息

由于需求中提到，要查询emp的所有数据，所以是不能内连接查询的，需要考虑使用外连接查询。

表结构：emp, dept

连接条件：emp.dept_id = dept.id

```
1  select e.*, d.name from emp e left outer join dept d on e.dept_id = d.id;
2
3  select e.*, d.name from emp e left join dept d on e.dept_id = d.id;
```

B. 查询dept表的所有数据，和对应的员工信息 (右外连接)

由于需求中提到，要查询dept表的所有数据，所以是不能内连接查询的，需要考虑使用外连接查询。

表结构：emp, dept

连接条件：emp.dept_id = dept.id

```
1  select d.*, e.* from emp e right outer join dept d on e.dept_id = d.id;
2
3  select d.*, e.* from dept d left outer join emp e on e.dept_id = d.id;
```

注意事项：

左外连接和右外连接是可以相互替换的，只需要调整在连接查询时SQL中，表结构的先后顺序就可以了。而我们在日常开发使用时，更偏向于左外连接。

5.5 自连接

5.5.1 自连接查询

自连接查询，顾名思义，就是自己连接自己，也就是把一张表连接查询多次。我们先来学习一下自连接的查询语法：

```
1  SELECT  字段列表  FROM  表A  别名A  JOIN  表A  别名B  ON  条件 ... ;
```

而对于自连接查询，可以是内连接查询，也可以是外连接查询。

案例：

A. 查询员工 及其 所属领导的名字

表结构：emp

```
1  select a.name , b.name from emp a , emp b where a.managerid = b.id;
```

B. 查询所有员工 emp 及其领导的名字 emp ，如果员工没有领导，也需要查询出来

表结构：emp a , emp b


```
1  select a.name '员工', b.name '领导' from emp a left join emp b on a.managerid =
   b.id;
```

注意事项：

在自连接查询中，必须要为表起别名，要不然我们不清楚所指定的条件、返回的字段，到底是哪一张表的字段。

5.5.2 联合查询

对于union查询，就是把多次查询的结果合并起来，形成一个新的查询结果集。

```
1  SELECT  字段列表  FROM  表A  ...
2  UNION [ ALL ]
3  SELECT  字段列表  FROM  表B  ....;
```

- 对于联合查询的多张表的列数必须保持一致，字段类型也需要保持一致。
- union all 会将全部的数据直接合并在一起，union 会对合并之后的数据去重。

案例：

A. 将薪资低于 5000 的员工，和 年龄大于 50 岁的员工全部查询出来。

当前对于这个需求，我们可以直接使用多条件查询，使用逻辑运算符 or 连接即可。那这里呢，我们也可以通过union/union all来联合查询。

```
1  select * from emp where salary < 5000
2  union all
3  select * from emp where age > 50;
```

id	name	age	job	salary	entrydate	managerid	dept_id
11	鹿杖客	56	职员	3750	2006-10-03	10	2
12	鹤笔翁	19	职员	3750	2007-05-09	10	2
15	俞莲舟	38	销售	4600	2004-10-12	14	4
16	宋远桥	40	销售	4600	2004-10-12	14	4
17	陈友谅	42	<null>	2000	2011-10-12	1	<null>
1	金庸	66	总裁	20000	2000-01-01	<null>	5
7	灭绝	60	财务总监	8500	2002-09-12	1	3
11	鹿杖客	56	职员	3750	2006-10-03	10	2
14	张三丰	88	销售总监	14000	2004-10-12	1	4

union all查询出来的结果，仅仅进行简单的合并，并未去重。

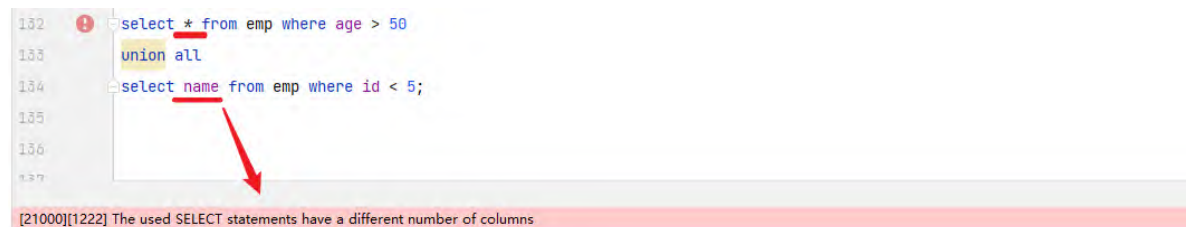
```
1 select * from emp where salary < 5000
2 union
3 select * from emp where age > 50;
```

id	name	age	job	salary	entrydate	managerid	dept_id
11	鹿杖客	56	职员	3750	2006-10-03	10	2
12	鹤笔翁	19	职员	3750	2007-05-09	10	2
15	俞莲舟	38	销售	4600	2004-10-12	14	4
16	宋远桥	40	销售	4600	2004-10-12	14	4
17	陈友谅	42	<null>	2000	2011-10-12	1	<null>
1	金庸	66	总裁	20000	2000-01-01	<null>	5
7	灭绝	60	财务总监	8500	2002-09-12	1	3
14	张三丰	88	销售总监	14000	2004-10-12	1	4

union 联合查询，会对查询出来的结果进行去重处理。

注意：

如果多条查询语句查询出来的结果，字段数量不一致，在进行union/union all联合查询时，将会报错。如：



5.6 子查询

5.6.1 概述

1). 概念

SQL语句中嵌套SELECT语句，称为嵌套查询，又称子查询。

```
1 SELECT * FROM t1 WHERE column1 = ( SELECT column1 FROM t2 );
```

子查询外部的语句可以是INSERT / UPDATE / DELETE / SELECT 的任何一个。

2). 分类

根据子查询结果不同，分为：

- A. 标量子查询 (子查询结果为单个值)
- B. 列子查询 (子查询结果为一列)
- C. 行子查询 (子查询结果为一行)
- D. 表子查询 (子查询结果为多行多列)

根据子查询位置，分为：

- A. WHERE之后
- B. FROM之后
- C. SELECT之后

5.6.2 标量子查询

子查询返回的结果是单个值 (数字、字符串、日期等)，最简单的形式，这种子查询称为标量子查询。

常用的操作符：= <> > >= < <=

案例：

- A. 查询 "销售部" 的所有员工信息

完成这个需求时，我们可以将需求分解为两步：

- ①. 查询 "销售部" 部门ID

```
1 select id from dept where name = '销售部';
```

- ②. 根据 "销售部" 部门ID，查询员工信息

```
1 select * from emp where dept_id = (select id from dept where name = '销售部');
```

- B. 查询在 "方东白" 入职之后的员工信息

完成这个需求时，我们可以将需求分解为两步：

- ①. 查询 方东白 的入职日期

```
1 select entrydate from emp where name = '方东白';
```

②. 查询指定入职日期之后入职的员工信息

```
1 select * from emp where entrydate > (select entrydate from emp where name = '方东白');
```

5.6.3 列子查询

子查询返回的结果是一列（可以是多行），这种子查询称为列子查询。

常用的操作符：IN 、 NOT IN 、 ANY 、 SOME 、 ALL

操作符	描述
IN	在指定的集合范围之内，多选一
NOT IN	不在指定的集合范围之内
ANY	子查询返回列表中，有任意一个满足即可
SOME	与ANY等同，使用SOME的地方都可以使用ANY
ALL	子查询返回列表的所有值都必须满足

案例：

A. 查询 "销售部" 和 "市场部" 的所有员工信息

分解为以下两步：

①. 查询 "销售部" 和 "市场部" 的部门ID

```
1 select id from dept where name = '销售部' or name = '市场部';
```

②. 根据部门ID，查询员工信息

```
1 select * from emp where dept_id in (select id from dept where name = '销售部' or name = '市场部');
```

B. 查询比 财务部 所有人工资都高的员工信息

分解为以下两步：

①. 查询所有 财务部 人员工资

```
1  select id from dept where name = '财务部';  
2  
3  select salary from emp where dept_id = (select id from dept where name = '财务部');
```

②. 比 财务部 所有人工资都高的员工信息

```
1  select * from emp where salary > all ( select salary from emp where dept_id =  
    (select id from dept where name = '财务部') );
```

C. 查询比研发部其中任意一人工资高的员工信息

分解为以下两步：

①. 查询研发部所有人工资

```
1  select salary from emp where dept_id = (select id from dept where name = '研发部');
```

②. 比研发部其中任意一人工资高的员工信息

```
1  select * from emp where salary > any ( select salary from emp where dept_id =  
    (select id from dept where name = '研发部') );
```

5.6.4 行子查询

子查询返回的结果是一行（可以是多列），这种子查询称为行子查询。

常用的操作符：= 、<> 、IN 、NOT IN

案例：

A. 查询与 "张无忌" 的薪资及直属领导相同的员工信息 ；

这个需求同样可以拆解为两步进行：

①. 查询 "张无忌" 的薪资及直属领导

```
1  select salary, managerid from emp where name = '张无忌';
```

②. 查询与 "张无忌" 的薪资及直属领导相同的员工信息 ；

```
1  select * from emp where (salary,managerid) = (select salary, managerid from emp
    where name = '张无忌');
```

5.6.5 表子查询

子查询返回的结果是多行多列，这种子查询称为表子查询。

常用的操作符：IN

案例：

A. 查询与 "鹿杖客" ， "宋远桥" 的职位和薪资相同的员工信息

分解为两步执行：

①. 查询 "鹿杖客" ， "宋远桥" 的职位和薪资

```
1  select job, salary from emp where name = '鹿杖客' or name = '宋远桥';
```

②. 查询与 "鹿杖客" ， "宋远桥" 的职位和薪资相同的员工信息

```
1  select * from emp where (job,salary) in ( select job, salary from emp where name =
    '鹿杖客' or name = '宋远桥' );
```

B. 查询入职日期是 "2006-01-01" 之后的员工信息 ， 及其部门信息

分解为两步执行：

①. 入职日期是 "2006-01-01" 之后的员工信息

```
1  select * from emp where entrydate > '2006-01-01';
```

②. 查询这部分员工， 对应的部门信息；

```
1  select e.*, d.* from (select * from emp where entrydate > '2006-01-01') e left
    join dept d on e.dept_id = d.id ;
```

5.7 多表查询案例

数据环境准备：

```
1  create table salgrade(  
2      grade int,  
3      losal int,  
4      hisal int  
5  ) comment '薪资等级表';  
6  
7  insert into salgrade values (1,0,3000);  
8  insert into salgrade values (2,3001,5000);  
9  insert into salgrade values (3,5001,8000);  
10 insert into salgrade values (4,8001,10000);  
11 insert into salgrade values (5,10001,15000);  
12 insert into salgrade values (6,15001,20000);  
13 insert into salgrade values (7,20001,25000);  
14 insert into salgrade values (8,25001,30000);
```

在这个案例中，我们主要运用上面所讲解的多表查询的语法，完成以下的12个需求即可，而这里主要涉及到的表就三张：emp员工表、dept部门表、salgrade薪资等级表。

1). 查询员工的姓名、年龄、职位、部门信息（隐式内连接）

表：emp , dept

连接条件：emp.dept_id = dept.id

```
1  select e.name , e.age , e.job , d.name from emp e , dept d where e.dept_id = d.id;
```

2). 查询年龄小于30岁的员工的姓名、年龄、职位、部门信息（显式内连接）

表：emp , dept

连接条件：emp.dept_id = dept.id

```
1  select e.name , e.age , e.job , d.name from emp e inner join dept d on e.dept_id =  
    d.id where e.age < 30;
```

3). 查询拥有员工的部门ID、部门名称

表: emp , dept

连接条件: emp.dept_id = dept.id

```
1  select distinct d.id , d.name from emp e , dept d where e.dept_id = d.id;
```

4). 查询所有年龄大于40岁的员工, 及其归属的部门名称; 如果员工没有分配部门, 也需要展示出来 (外连接)

表: emp , dept

连接条件: emp.dept_id = dept.id

```
1  select e.*, d.name from emp e left join dept d on e.dept_id = d.id where e.age > 40 ;
```

5). 查询所有员工的工资等级

表: emp , salgrade

连接条件 : emp.salary >= salgrade.losal and emp.salary <= salgrade.hisal

```
1  -- 方式一
2  select e.* , s.grade , s.losal, s.hisal from emp e , salgrade s where e.salary >= s.losal and e.salary <= s.hisal;
3  -- 方式二
4  select e.* , s.grade , s.losal, s.hisal from emp e , salgrade s where e.salary between s.losal and s.hisal;
```

6). 查询 "研发部" 所有员工的信息及 工资等级

表: emp , salgrade , dept

连接条件 : emp.salary between salgrade.losal and salgrade.hisal ,

emp.dept_id = dept.id

查询条件 : dept.name = '研发部'

```
1  select e.* , s.grade from emp e , dept d , salgrade s where e.dept_id = d.id and ( e.salary between s.losal and s.hisal ) and d.name = '研发部';
```


7). 查询 "研发部" 员工的平均工资

表: emp , dept

连接条件 : emp.dept_id = dept.id

```
1  select avg(e.salary) from emp e, dept d where e.dept_id = d.id and d.name = '研发部';
```

8). 查询工资比 "灭绝" 高的员工信息。

①. 查询 "灭绝" 的薪资

```
1  select salary from emp where name = '灭绝';
```

②. 查询比她工资高的员工数据

```
1  select * from emp where salary > ( select salary from emp where name = '灭绝' );
```

9). 查询比平均薪资高的员工信息

①. 查询员工的平均薪资

```
1  select avg(salary) from emp;
```

②. 查询比平均薪资高的员工信息

```
1  select * from emp where salary > ( select avg(salary) from emp );
```

10). 查询低于本部门平均工资的员工信息

①. 查询指定部门平均薪资

```
1  select avg(e1.salary) from emp e1 where e1.dept_id = 1;  
2  select avg(e1.salary) from emp e1 where e1.dept_id = 2;
```

②. 查询低于本部门平均工资的员工信息

```
1  select * from emp e2 where e2.salary < ( select avg(e1.salary) from emp e1 where  
e1.dept_id = e2.dept_id );
```

11). 查询所有的部门信息，并统计部门的员工人数

```
1  select d.id, d.name , ( select count(*) from emp e where e.dept_id = d.id ) '人数'
   from dept d;
```

12). 查询所有学生的选课情况，展示出学生名称，学号，课程名称

表: student , course , student_course

连接条件: student.id = student_course.studentid , course.id = student_course.courseid

```
1  select s.name , s.no , c.name from student s , student_course sc , course c where
   s.id = sc.studentid and sc.courseid = c.id ;
```

备注：以上需求的实现方式可能会很多，SQL写法也有很多，只要能满足我们的需求，查询出符合条件的记录即可。

6. 事务

6.1 事务简介

事务 是一组操作的集合，它是一个不可分割的工作单位，事务会把所有的操作作为一个整体一起向系统提交或撤销操作请求，即这些操作要么同时成功，要么同时失败。

就比如：张三给李四转账1000块钱，张三银行账户的钱减少1000，而李四银行账户的钱要增加1000。这一组操作就必须在一个事务的范围内，要么都成功，要么都失败。



正常情况：转账这个操作，需要分为以下这么三步来完成，三步完成之后，张三减少1000，而李四增加1000，转账成功：

- 查询张三账户余额
- 张三账户余额-1000
- 李四账户余额+1000

id	name	money
1	张三	1000
2	李四	3000

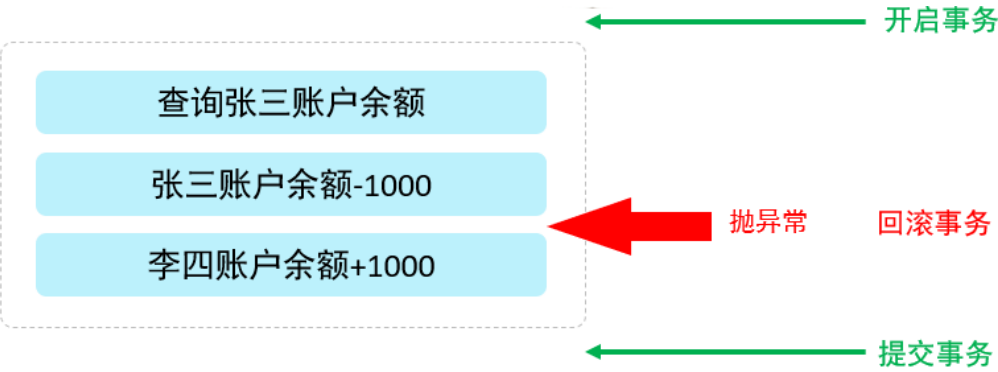
异常情况：转账这个操作，也是分为以下这么三步来完成，在执行第三步是报错了，这样就导致张三减少1000块钱，而李四的金额没变，这样就造成了数据的不一致，就出现问题了。

- 查询张三账户余额
- 张三账户余额-1000
- 李四账户余额+1000

抛异常

id	name	money
1	张三	1000
2	李四	2000

为了解决上述的问题，就需要通过数据的事务来完成，我们只需要在业务逻辑执行之前开启事务，执行完毕后提交事务。如果执行过程中报错，则回滚事务，把数据恢复到事务开始之前的状态。



注意：默认MySQL的事务是自动提交的，也就是说，当执行完一条DML语句时，MySQL会立即隐式的提交事务。

6.2 事务操作

数据准备：

```

1  drop table if exists account;
2
3  create table account(
4      id int primary key AUTO_INCREMENT comment 'ID',
5      name varchar(10) comment '姓名',
6      money double(10,2) comment '余额'
7  ) comment '账户表';
8
9  insert into account(name, money) VALUES ('张三',2000), ('李四',2000);

```

6.2.1 未控制事务

1). 测试正常情况

```

1  -- 1. 查询张三余额
2  select * from account where name = '张三';
3
4  -- 2. 张三的余额减少1000
5  update account set money = money - 1000 where name = '张三';
6
7  -- 3. 李四的余额增加1000
8  update account set money = money + 1000 where name = '李四';

```

测试完毕之后检查数据的状态，可以看到数据操作前后是一致的。

id	name	money
1	张三	1000
2	李四	3000




2). 测试异常情况

```

1  -- 1. 查询张三余额
2  select * from account where name = '张三';
3
4  -- 2. 张三的余额减少1000
5  update account set money = money - 1000 where name = '张三';
6  出错了....
7
8  -- 3. 李四的余额增加1000
9  update account set money = money + 1000 where name = '李四';

```

我们把数据都恢复到2000， 然后再次一次性执行上述的SQL语句(出错了.... 这句话不符合SQL语法, 执行就会报错)， 检查最终的数据情况， 发现数据在操作前后不一致了。

 id	 name	 money
1	张三	1000
2	李四	2000

6.2.2 控制事务一

1). 查看/设置事务提交方式

```
1  SELECT  @@autocommit ;
2  SET     @@autocommit = 0 ;
```

2). 提交事务

```
1  COMMIT;
```

3). 回滚事务

```
1  ROLLBACK;
```

注意：上述的这种方式，我们是修改了事务的自动提交行为，把默认的自动提交修改为了手动提交，此时我们执行的DML语句都不会提交，需要手动的执行commit进行提交。

6.2.3 控制事务二

1). 开启事务

```
1  START TRANSACTION  或  BEGIN ;
```

2). 提交事务

```
1  COMMIT;
```

3). 回滚事务

```
1    ROLLBACK;
```

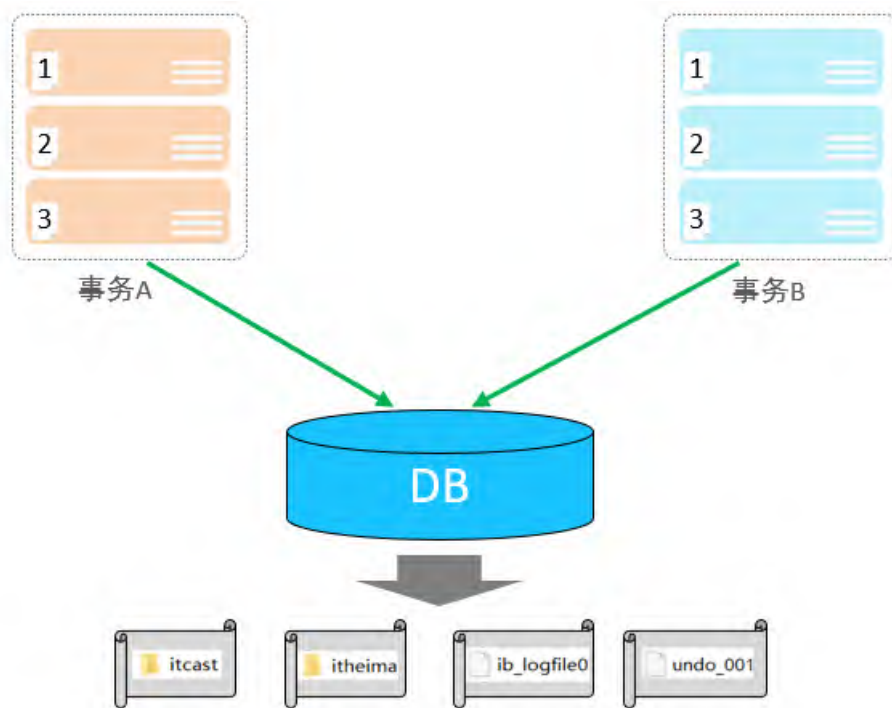
转账案例：

```
1    -- 开启事务
2    start transaction
3
4    -- 1. 查询张三余额
5    select * from account where name = '张三';
6
7    -- 2. 张三的余额减少1000
8    update account set money = money - 1000 where name = '张三';
9
10   -- 3. 李四的余额增加1000
11   update account set money = money + 1000 where name = '李四';
12
13   -- 如果正常执行完毕，则提交事务
14   commit;
15   -- 如果执行过程中报错，则回滚事务
16   -- rollback;
```

6.3 事务四大特性

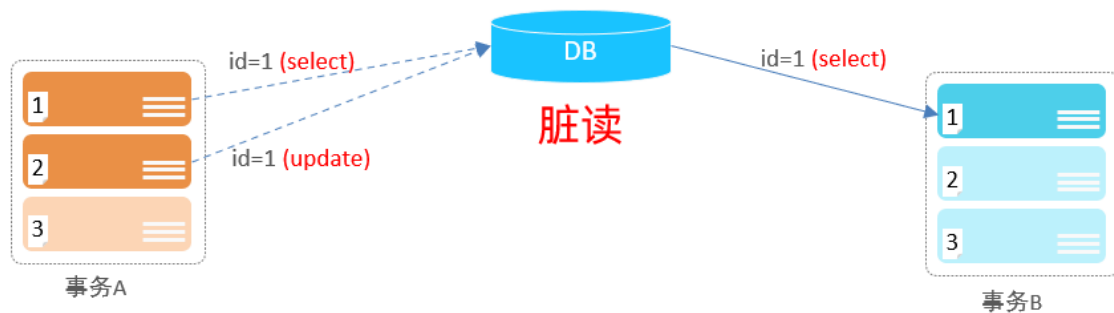
- 原子性 (Atomicity)：事务是不可分割的最小操作单元，要么全部成功，要么全部失败。
- 一致性 (Consistency)：事务完成时，必须使所有的数据都保持一致状态。
- 隔离性 (Isolation)：数据库系统提供的隔离机制，保证事务在不受外部并发操作影响的独立环境下运行。
- 持久性 (Durability)：事务一旦提交或回滚，它对数据库中的数据的改变就是永久的。

上述就是事务的四大特性，简称ACID。



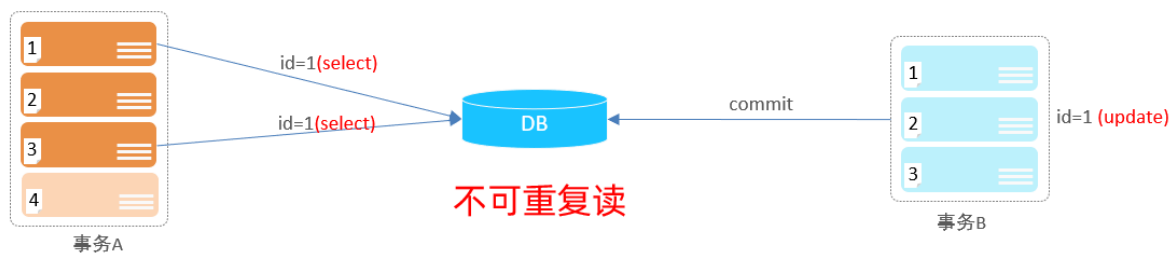
6.4 并发事务问题

1). 脏读：一个事务读到另外一个事务还没有提交的数据。



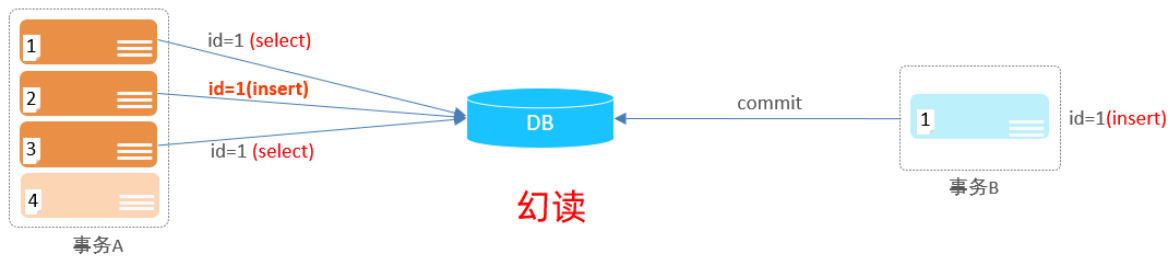
比如B读取到了A未提交的数据。

2). 不可重复读：一个事务先后读取同一条记录，但两次读取的数据不同，称之为不可重复读。



事务A两次读取同一条记录，但是读取到的数据却是不一样的。

3). 幻读：一个事务按照条件查询数据时，没有对应的数据行，但是在插入数据时，又发现这行数据已经存在，好像出现了 "幻影"。



6.5 事务隔离级别

为了解决并发事务所引发的问题，在数据库中引入了事务隔离级别。主要有以下几种：

隔离级别	脏读	不可重复读	幻读
Read uncommitted	√	√	√
Read committed	×	√	√
Repeatable Read (默认)	×	×	√
Serializable	×	×	×

1). 查看事务隔离级别

```
1 SELECT @@TRANSACTION_ISOLATION;
```

2). 设置事务隔离级别

```
1 SET [ SESSION | GLOBAL ] TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED |  
    READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

注意：事务隔离级别越高，数据越安全，但是性能越低。

