## MySQL数据库

**课程大纲**

**一、MySQL数据库**

**二、SQLyog与Navicat客户端**

**三、SQL编程语言**

**四、数据类型**

**五、约束类型**

**六、视图、存储过程、函数、事务、索引**

### 一、课程介绍

#### 1、为什么学习数据库？

**数据库是软件的基础。**

大部分软件都需要操作数据，比如游戏、社交、新闻、商城、财务等，这些软件都在不停的展示、存储数据，它们的数据都存储在数据库，数据库是软件的基础。



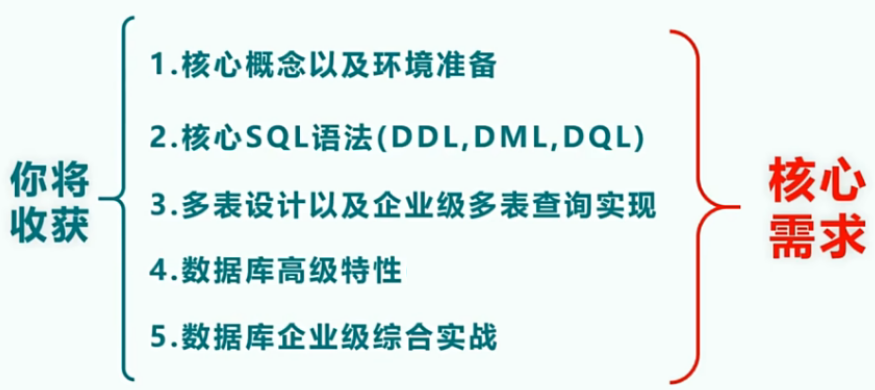
测试工程师在测试软件的过程中，不仅需要在界面上操作，还需要检查数据库中的数据是否正确？从而在软件出问题时，测出更深层的问题。



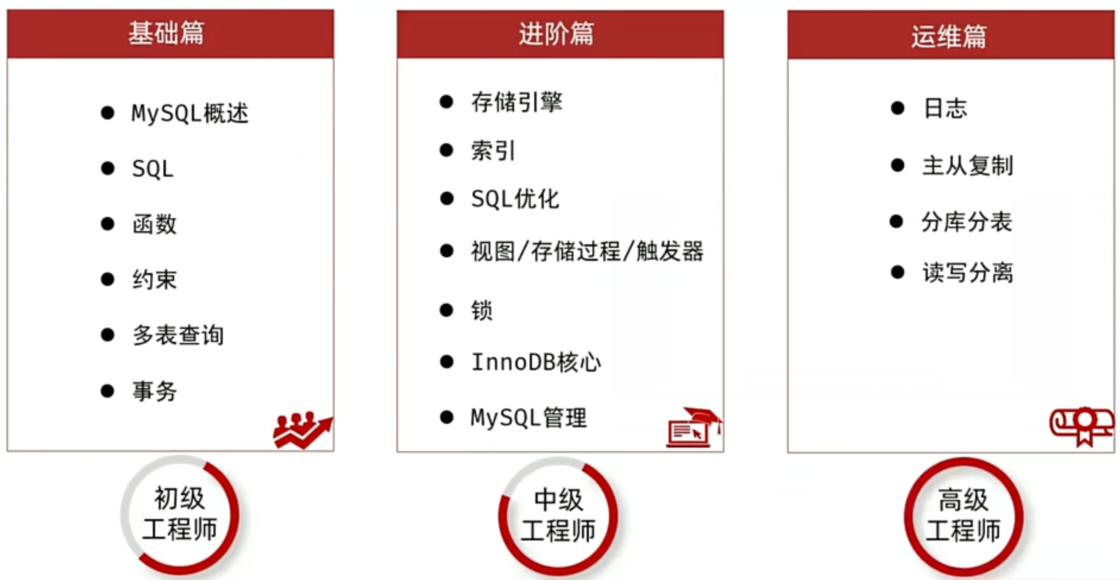
如：测试注册登录功能，在输入了注册信息后，提示注册成功，但是使用刚才注册的信息登录不成功。 这时需要检查数据库中是否保存了正确的注册信息，如果数据库中没有保存数据，那么使用刚才注册的账号肯定登录不了，这样就能定位问题：注册时，没有把数据存储起来。

#### 2、课程内容





MySQL学习进阶路线：



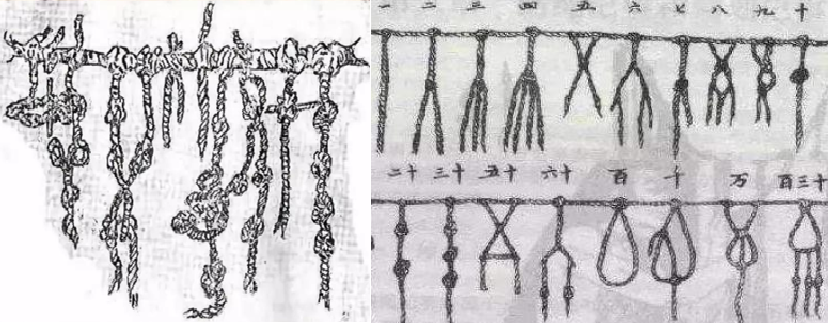
### 二、初始数据库

#### 1、了解数据存储的发展阶段

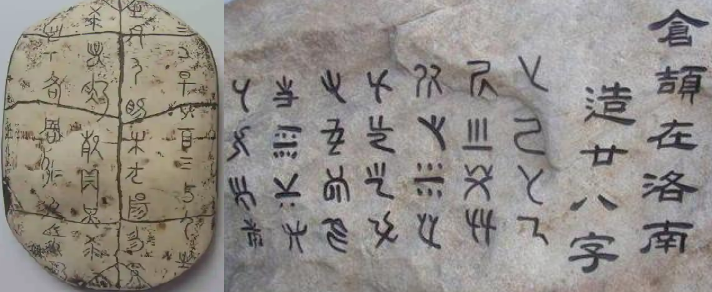
**1.1人工管理阶段**

人工管理阶段，数据都是靠人工进行整理和保存的，使用起来很不方便。不便于查询、共享、保存。

结绳记事：



甲骨文字：



纸：

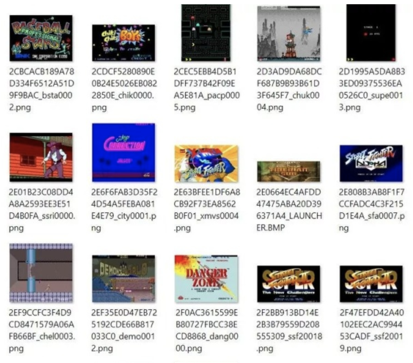


**1.2 文件系统阶段**

文件系统阶段，将数据存储在磁盘上，数据都是以文件的形式出现，通过文件系统管理。

相对于人工管理阶段方便的多，但是在查询数据内容时，还不是很方便。

磁盘：



**1.3 数据库系统阶段**

**数据库：**是指长期存储在计算机内、有组织的数据集合。简而言之，数据库就是一个存储数据的仓库。

如：文字、图片、音频、视频 ......

**表：**是数据库中存储数据的基本单位，数据按照分类存储到不同的表中，能够非常高效的查询其中的数据。

**优点：**

**使数据永久储存**（DB工程师，维护数据库）。

**实现数据共享**（一个数据多个人同时访问）。

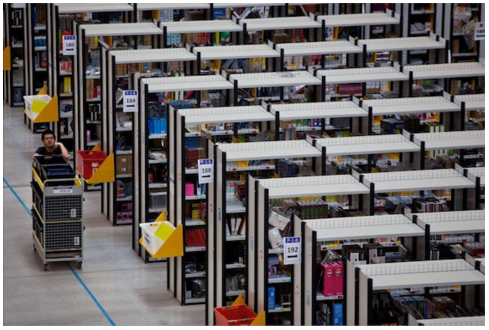
网页中显示的商品信息：



数据库中存储的商品数据：



真实的商品：



#### 2、相关概念

**数据库：**

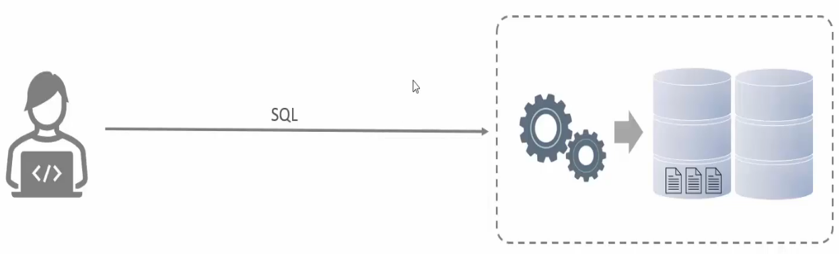
存储数据的仓库，数据是有组织的进行存储。

**数据库管理系统：**

操纵和管理数据库的大型软件。

**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。



**主流的关系型数据库系统：**

Oracle：在大型项目中使用，例如：银行、电信等项目；

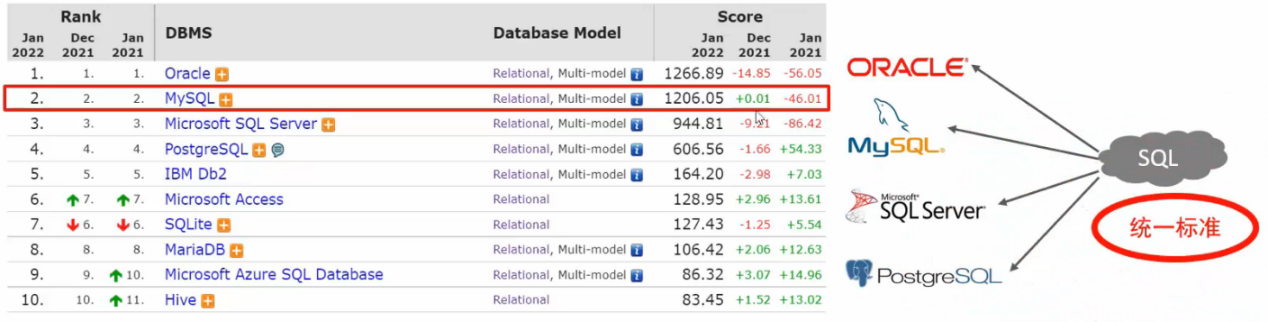
MySQL：Web 项目中使用最广泛的关系型数据库；

Microsoft SQL Server：在微软的项目中使用；

SQLite：轻量级数据库，主要应用在移动平台；

还有非关系型数据库，如：MongDB、 DB2 ......

数据库排名：https://db-engines.com/en/ranking



**数据库的核心元素：**

**库**：数据表的集合，一个数据库中可以有多个表。

**表**：数据行的集合，一个表中可以有多个列。

**列**：字段。

**行**：一条记录。

**总结：**

1.数据库：数据存储的仓库。（库、表、列、行）

永久存储，数据共享，高效，......

2.数据库管理系统：操纵和管理数据库的大型软件。

3.SQL：操作关系型数据库的编程语言，也是一套标准。（通用）

### 三、MySQL数据库

#### 1、MySQL介绍

MySQL是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 旗下产品。

MySQL是最流行的关系型数据库管理系统之一。

MySQL前身是由瑞典 MySQL AB 公司开发的数据库；

2008年被 Sun 公司收购；

2009年 Sun 公司又被 Oracle 公司收购。

**MySQL特点：**

开源，免费，安装体积小，易安装，使用率广泛，......

**版本：**MySQL官方提供了两种不同的版本。

**社区版**(MySQL Community Server)：

免费，MySQL不提供任何技术支持。

**商业版**(MySQL Enterprise Edition)：

收费，可以试用30天，官方提供技术支持。

本课程采用的是社区版（MySQL Community Server 5.7.9）。

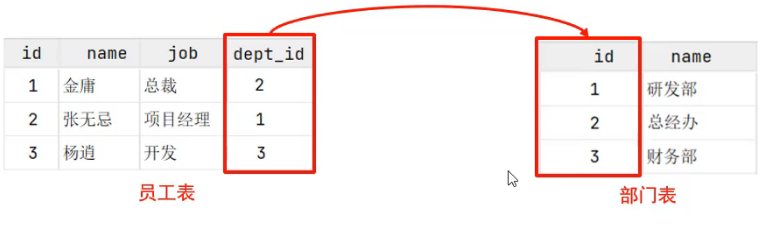
**关系型数据库(RDBMS)：**

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。

**特点:**

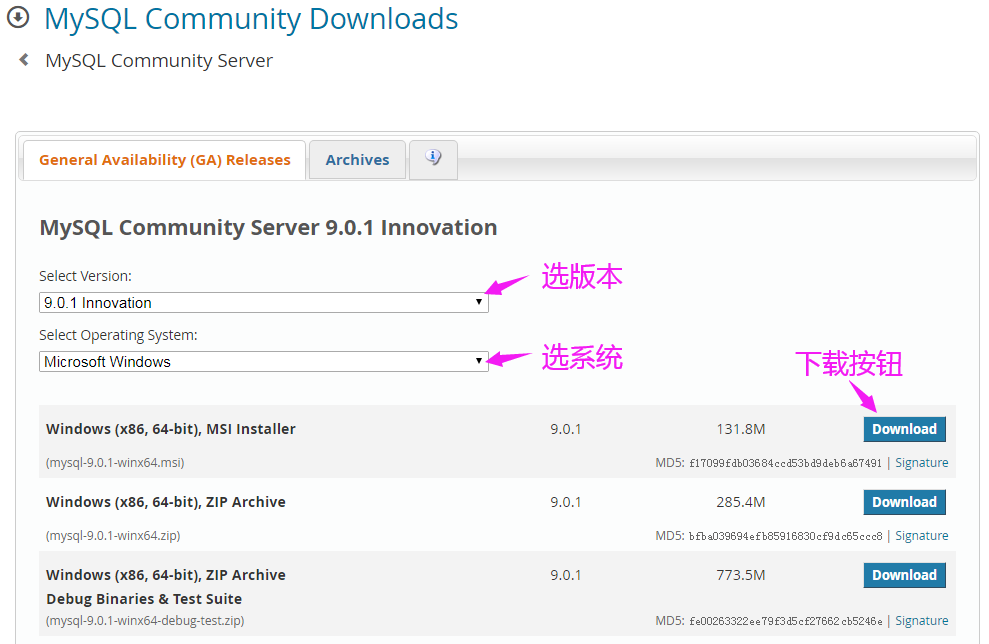
1.使用表存储数据， 格式统一，便于维护。

2.使用SQL语言操作，标准统一，使用方便。



#### 2、MySQL下载

下载地址：https://dev.mysql.com/downloads/mysql/

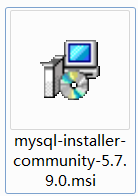


#### 3、MySQL安装

相关安装教程：（百度搜索“MySQL安装教程”）

https://www.cnblogs.com/acmexyz/p/10076896.html

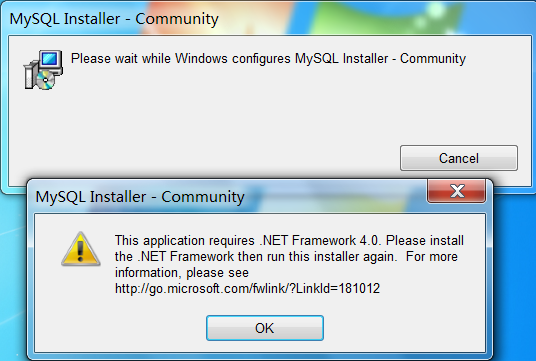
**双击MySQL安装包按照教程安装即可。**



#### 4、异常处理

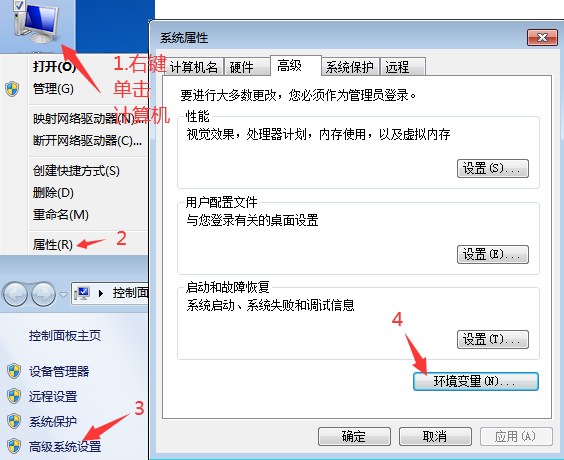
若出现以下提示表示.net缺失，

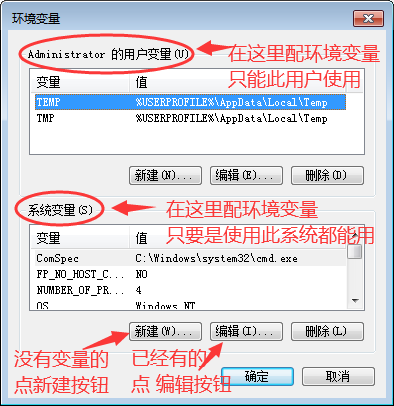
需先安装.net4.0以上版本后才能安装MySQL。



#### 5、配置MySQL环境

电脑右键-->属性-->高级系统设置-->环境变量-->系统变量

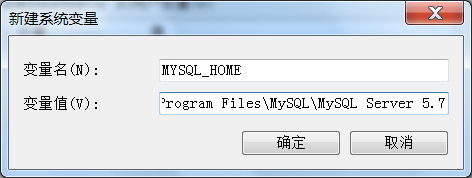




**第一步：新建系统变量**

**变量名**：MYSQL\_HOME

**变量值**：C:\Program Files\MySQL\MySQL Server 5.7



注意：**变量值**是MySQL的安装**路径**，看到bin目录即可。

**第二步：编辑Path系统变量**

**变量值**：%MYSQL\_HOME%\bin;





注意：这段命令行放在变量值的最前端，连;也带上，一个标点符号都不能少。

#### 6、管理MySQL服务

**第一种：通过DOS命令实现启动/停止MySQL服务。**

net start 服务名 启动 MySQL服务。

如：net start mysql57

表示启动mysql57 服务（本机的服务名是mysql57）。

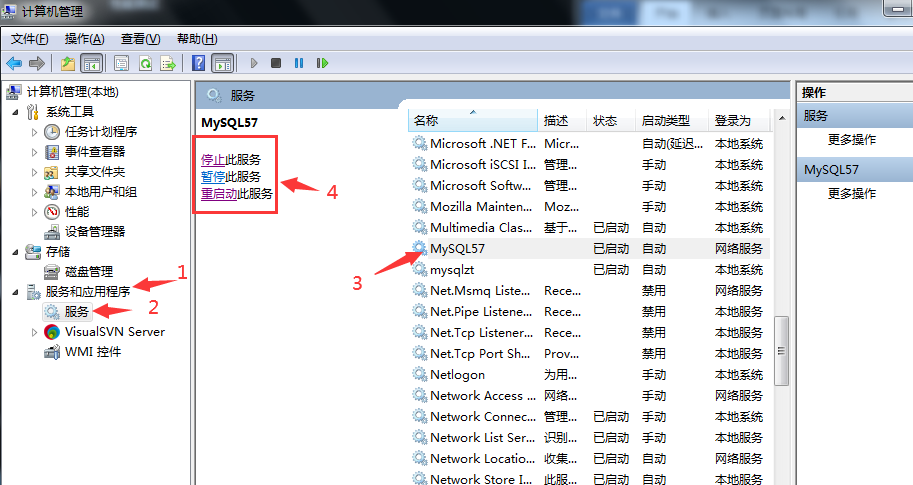
net stop 服务名 停止 MySQL服务。

如：net stop mysql57

表示停止mysql57 服务。

**第二种：通过windows服务管理实现 启动/停止 MySQL服务。**

右键单击计算机-->管理-->服务和应用程序-->服务-->数据库服务（本机的服务名是mysql57）-->选择服务左侧上角操作按钮（停止，启动，重启）。



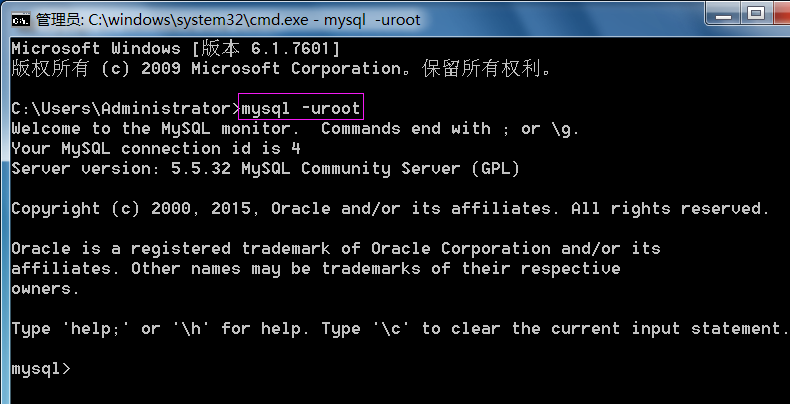
#### 7、访问MySQL数据库

**7.1 通过DOS命令访问MySQL数据库**

mysql -uroot -p密码

如：mysql -uroot -p 123456

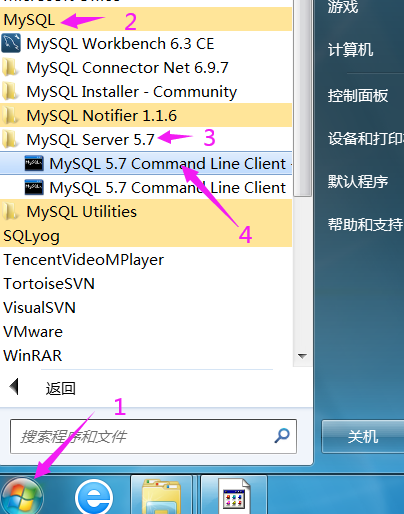


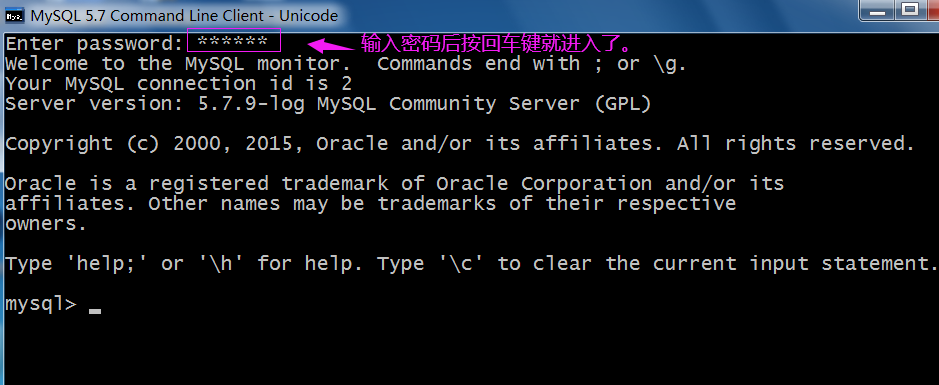


**7.2 通过mysql自带的客服端访问MySQL数据库**

操作步骤：

左下角win图标-->mysql-->mysql server 5.7-->MySQL 5.7 Command Line Client-->输入密码-->回车





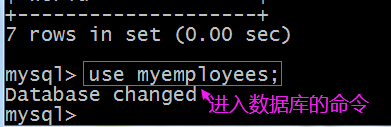
**7.3 相关查看命令**

show databases; 表示查看MySQL数据库中所有的数据库名称。

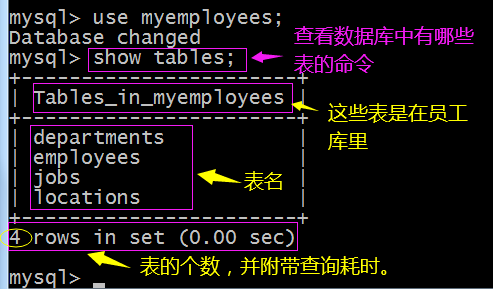


use 库名；

如 use myemployees; 表示进入该数据库。



show tables; 表示查看当前库中有哪些表。

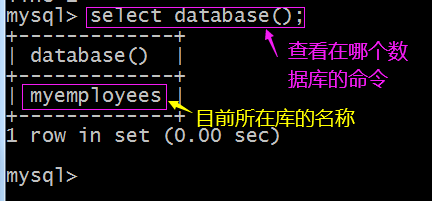


select \* from 表名；

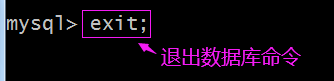
如 select \* from jobs; 表示查看目标表中所有数据。



select database()； 表示查看当前在哪个库。



exit； 表示退出数据库。



### 四、客户端安装与使用

#### 1、客户端下载与安装

客户端可视化工具有：**sqlyog 、navicat**

**下载：**在电脑软件管家中搜索客户端名称即可下载。

**安装：**双击打开安装包后，按提示下一步即可完成安装。



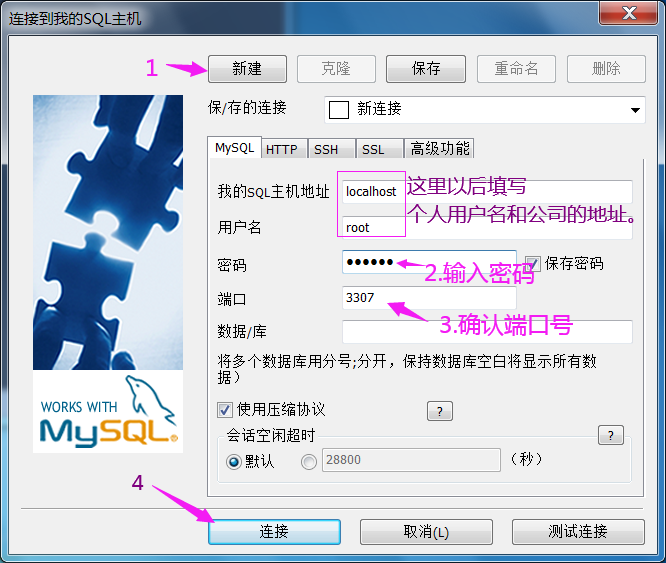
#### 2、SQLyog

**名称**: any

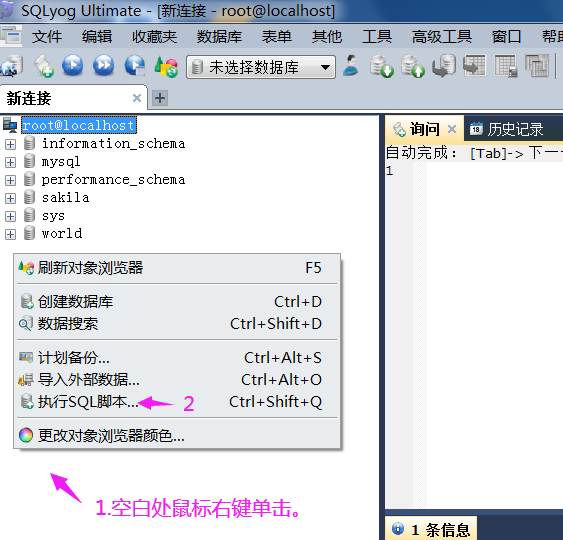
**秘钥**: dd987f34-f358-4894-bd0f-21f3f04be9c1



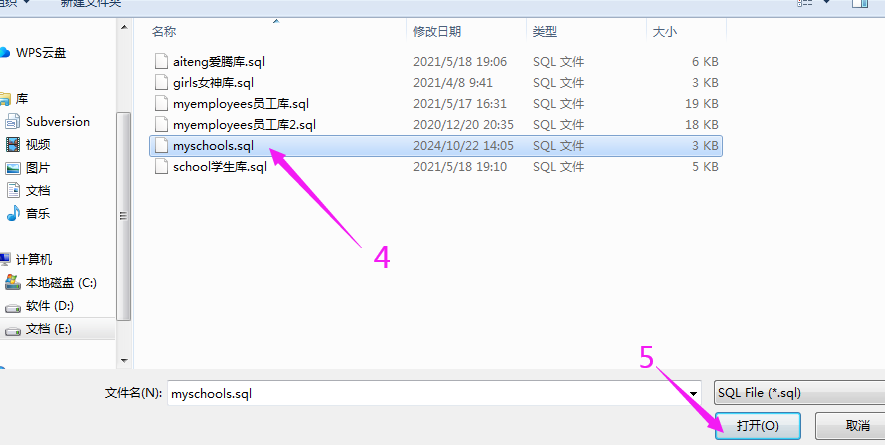
**sqlyog （客户端）连接 MySQL（服务端）步骤：**

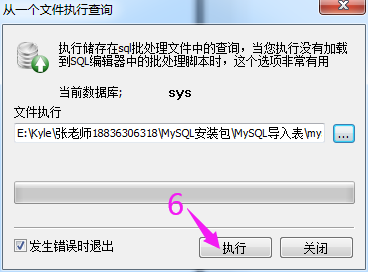


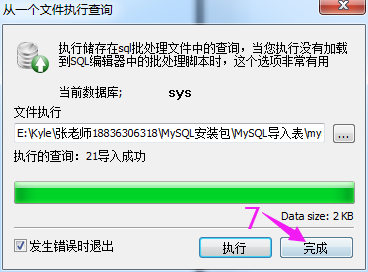
**sqlyog里执行sql脚本文件步骤：**

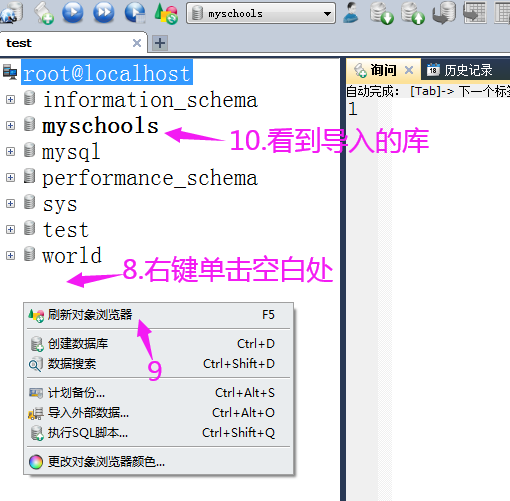




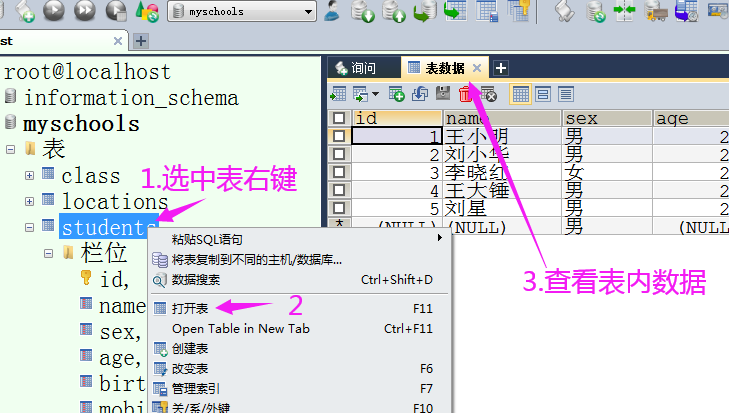




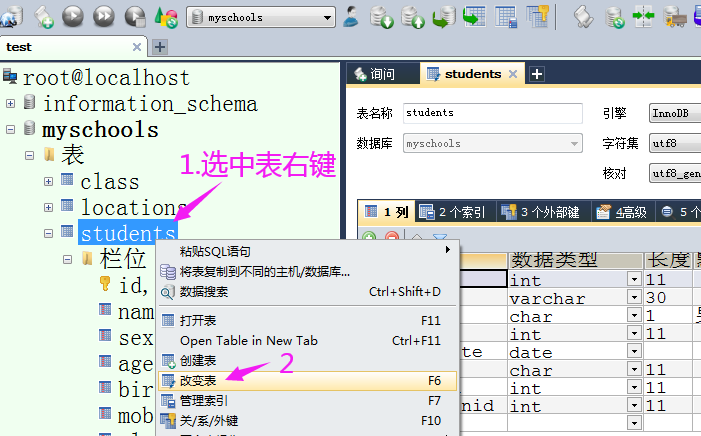




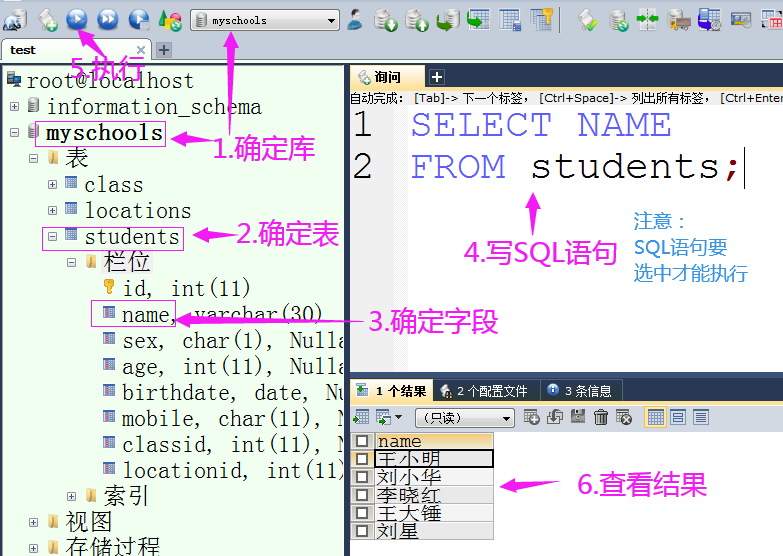
**查看表中数据：**



**查看表中各字段信息：**



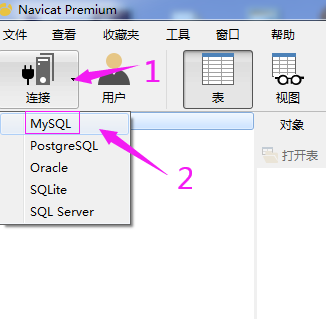
**SQLyog 中编写并执行SQL语句：**

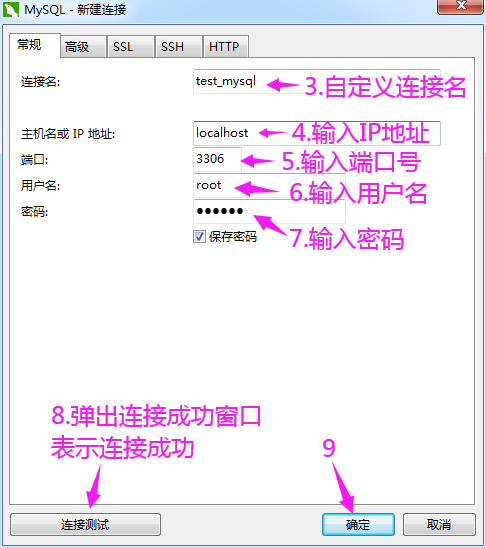


#### 3、Navicat

Navicat（客户端）连接 MySQL（服务端）

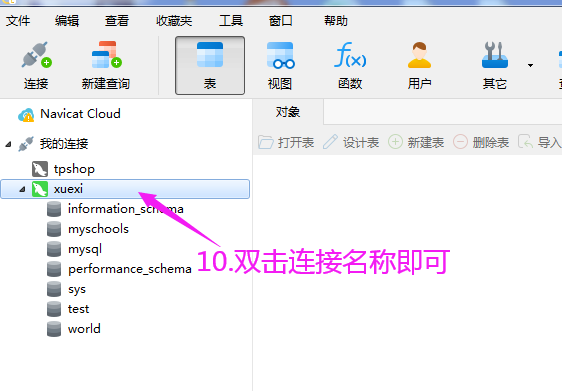
打开 Navicat ，点击连接 ，选择 MySQL。



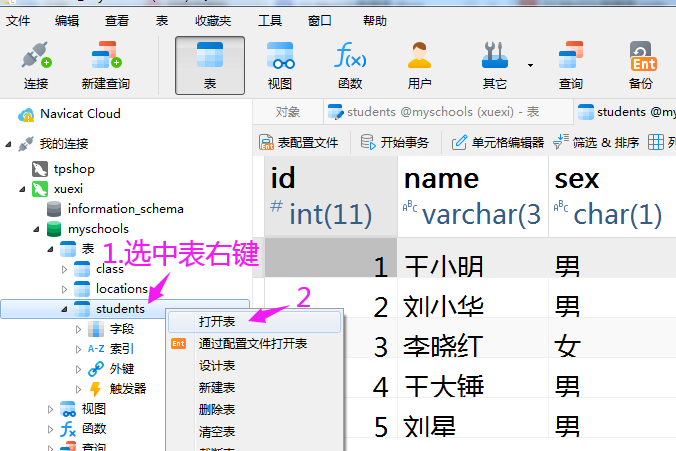


**双击**建立好的连接名：

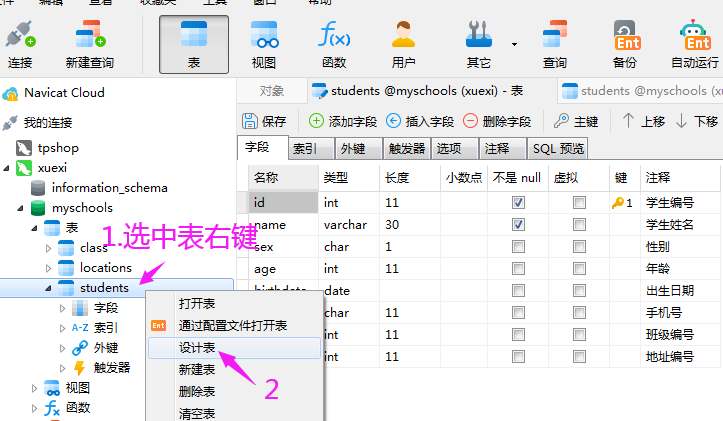
**绿色**表示连接成功！**灰色**表示未连接。



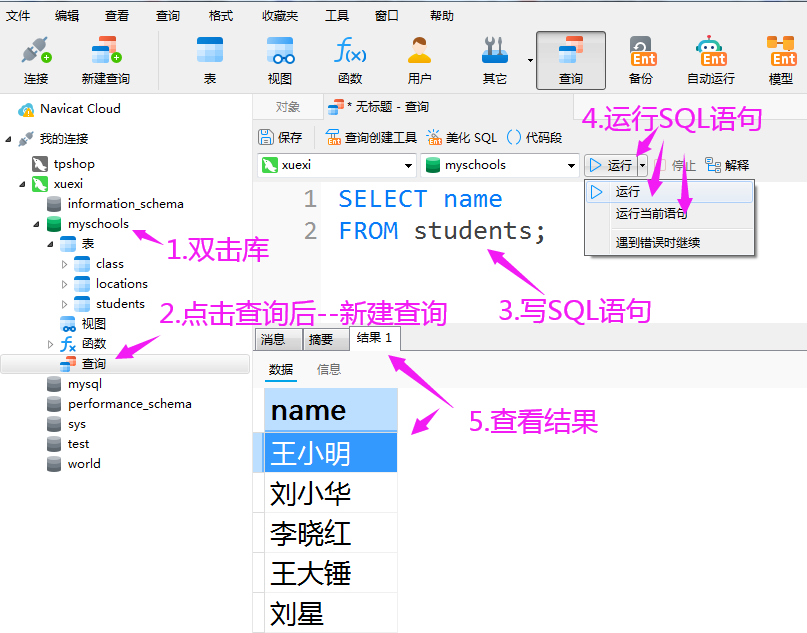
**查看表中数据：**



**查看表中各字段信息：**



**Navicat 中编写并执行SQL语句：**

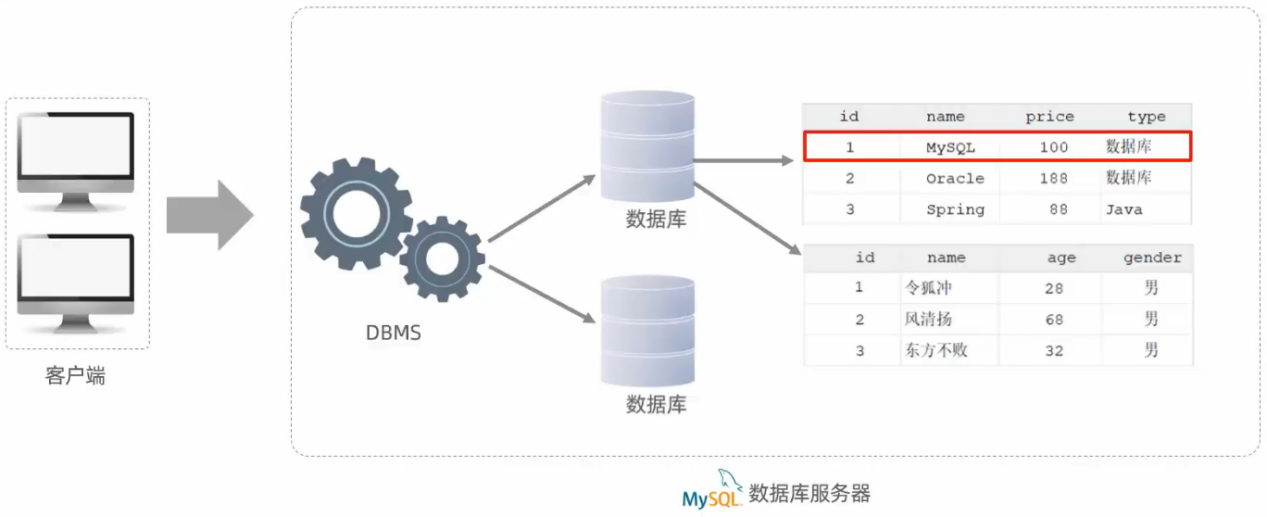


**客户端与服务端搭配场景:**

在实际工作中，数据库的服务端是部署在服务器上的，我们要想使用数据库，就需要使用客户端远程连接数据库服务器。

客户端**远程**连接服务端（工作场景）。

客户端**本地**连接服务端（教学场景）。



### 五、SQL语言介绍

Structure Query Language(结构化查询语言)简称SQL，它被美国国家标准局(ANSI)确定为关系型数据库语言的美国标准，后被国际化标准组织(ISO)采纳为关系数据库语言的国际标准。数据库管理系统可以通过SQL语言管理数据库、定义和操作数据，维护数据的完整性和安全性。

**SQL语言的优点：**简单、易学、通用、......

#### 1、SQL通用语法

SQL语句可以单行或多行书写，以分号结尾；

SQL语句可以使用空格缩进来增强语句的可读性；

MySQL数据库的SQL语句**不区分大小写**，关键字建议使用大写。

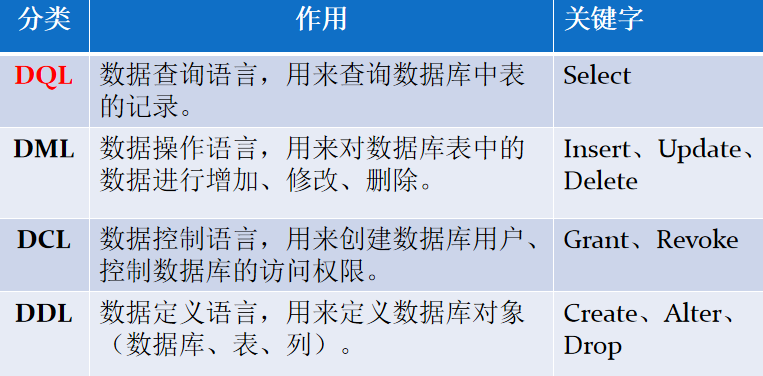
**注释:**

加了注释的语句不会被程序执行。

单行注释: -- 注释内容 或 # 注释内容(MySQL特有)

多行注释: /\* 注释内容 \*/

#### 2、SQL语言分类



对于测试工程师来说，重点是数据的查询，因此需要熟练编写**DQL**，其它类型，如 DCL、DDL、DML了解即可。

### 六、DQL数据查询语言

DQL语言主要用来查询数据库表中的记录。有单表查询、多表查询、子查询等语法。

#### 1、单表查询

**select 字段列表**

**from 表名**

**where 条件列表**

**group by 分组字段列表**

**having 分组后条件列表**

**order by 排序字段列表**

**limit 分页参数**

##### 1.1 基本查询

查询一个字段语法：（select，from）

select 字段 from 表名；

查询多个字段语法：

select 字段1，字段2，字段3，... from 表名；

查询所有字段语法：

select \* from 表名；

设置别名语法：

select 字段1[as 别名1]，字段2[as 别名2]，... from 表名；

去重语法：

select distinct 字段 from 表名；

**注意事项：**

通常将表中的列名称之为字段；

查询的字段必须是表中已有的字段；

\*号代表所有字段（尽量少用，会影响效率）；

字段与字段之间用逗号隔开，最后一个字段无需加逗号；

所有符号必须是英文输入法的符号；

SQL语句可以多行书写，以分号结束；

若SQL语句中有字符串出现，则必须加引号。

**结果集：**

第一行是表头（字段/列名）；

从第二行开始，才是要查询的数据。

**基本查询练习：**

# 查询学生信息（所有字段）

SELECT \* FROM students;

# 查询学生姓名（一个字段）

SELECT NAME FROM students;

# 查询学生姓名、性别、手机号（多个个字段）

SELECT NAME,sex,mobile FROM students;

# 查询学生姓名、性别，要求：起别名（设置别名）

SELECT NAME AS 姓名,sex AS 性别 FROM students;

# 查询学生年龄，要求：年龄去重（去重）

SELECT DISTINCT age FROM students;

**# 课后练习**

# 1、MySQL的特点是什么？

# 2、如何安装sqlyog，Navicat？

# 3、如何使用sqlyog，Navicat连接MySQL？

# 4、如何操作sqlyog，Navicat？

# 5、基本查询的用法？

##### 1.2 条件查询

条件查询语法：（where）

select 字段列表 from 表名 where 条件列表；

使用 where 子句对表中的数据筛选，符合条件的数据会出现在结果集中。

在条件查询中常见的运算符有比较运算符和逻辑运算符等多种筛选方式。

**比较运算符：**

> 大于

< 小于

>= 大于等于

<= 小于等于

= 等于

!= 或 <> 不等于

in(...) 在in之后的列表中的值

between...and... 在某个范围之内（含最小、最大值）

like‘通配符’ 模糊匹配

is null 是null（null 为空值）

**逻辑运算符：**

and 并且（多个条件，同时成立）

or 或者（多个条件，任意一个成立）

not 非（不是，取反的意思）

**注意事项：**

Null值通常不能参与运算，需要特殊处理。

通配符：\_匹配单个字符，%匹配任意个字符。

转义符：\将后面的通配符转义为普通字符。

**条件查询练习：**

# 查询年龄为23岁的学生信息

SELECT \* FROM students WHERE age=23;

# 查询女生信息

SELECT \* FROM students WHERE sex='女';

# 查询年龄大于23岁的学生信息

SELECT \* FROM students WHERE age>23;

# 查询年龄小于23岁的学生信息

SELECT \* FROM students WHERE age<23;

# 查询年龄不是23岁的学生信息

SELECT \* FROM students WHERE age!=23;

SELECT \* FROM students WHERE NOT age=23;

# 查询姓王的学生信息

SELECT \* FROM students WHERE NAME LIKE'王%';

# 查询手机号码包含7的学生信息

SELECT \* FROM students WHERE mobile LIKE'%7%';

# 查询手机号码为空的学生信息

SELECT \* FROM students WHERE mobile IS NULL;

# 查询手机号码不为空的学生信息

SELECT \* FROM students WHERE NOT mobile IS NULL;

# 查询年龄为23岁的男生信息

SELECT \* FROM students WHERE age=23 AND sex='男';

# 查询年龄为23岁或25岁的学生信息

SELECT \* FROM students WHERE age=23 OR age=25;

SELECT \* FROM students WHERE age IN(23,25);

# 查询年龄22到25岁的学生信息

SELECT \* FROM students WHERE age>=22 AND age<=25;

SELECT \* FROM students WHERE age BETWEEN 22 AND 25;

**扩展知识：AND与OR的优先级。**

SQL语句中，若需要组合使用AND和OR时，为了明确操作符的优先级（AND的优先级高于OR），建议使用括号()来明确指定条件的组合方式。

# 查询性别为男或女，并且年龄为22岁的学生信息

SELECT \*

FROM students

WHERE sex='男' OR sex='女' AND age=22 ;

SELECT \*

FROM students

WHERE (sex='男' OR sex='女') AND age=22 ;

##### 1.3 聚合函数查询

MySQL内置了很多种函数，聚合函数只是其中一种。

所谓聚合函数，就是将一列数据作为一个整体，进行纵向计算。

常用的聚合函数如下：

sum() 求和

avg() 求平均值

max() 求最大值

min() 求最小值

count() 统计数量

聚合函数语法：

select聚合函数(字段列表) from 表名;

**注意事项：**

聚合函数不对null值进行运算。

聚合函数不能在 where 命令中使用。

**count(\*)与count(字段)的区别：**

count(\*)时不过滤null值；count(字段)时会过滤null值。

**聚合函数练习：**

# 查询学生的年龄

SELECT age FROM students;

# 查询学生的年龄总和

SELECT SUM(age) FROM students;

# 查询学生的平均年龄

SELECT AVG(age) FROM students;

# 查询学生的最大年龄

SELECT MAX(age) FROM students;

# 查询学生的最小年龄

SELECT MIN(age) FROM students;

# 查询学生人数

SELECT COUNT(\*) FROM students;

SELECT COUNT(id) FROM students;

# 查询学生表的班级数量

SELECT COUNT( DISTINCT classid) FROM students;

##### 1.4 分组查询

分组查询语法格式：（group by，having）

select 字段，聚合函数

from 表名

[where 条件]

group by 分组字段列表

having 分组后筛选条件;

group by是把其命令后的字段值进行分组；

也就是把该字段内相同的数据分到一个组中，以便于对每组数据进行统计。

having是筛选聚合函数时常用的命令。

**注意事项：**

**1.分组条件：**关键字 + 聚合函数

通常需求若出现关键字并且还有聚合函数的时候考虑使用分组查询；

关键字：“哪个”、“每个”、 “各个”等。（关键字有可能不显示在需求中）；

聚合函数：SUM,AVG,MAX,MIN,COUNT。

**2.数据失真：**分组之后，查询的字段建议是分组字段或聚合函数；查询其他字段无意义，还有可能导致数据失真。

分组字段？GROUP BY 命令后的字段（判断技巧：关键字后面的字段）。

**3.分组多个字段：**GROUP BY 后可以跟多个字段，字段与字段之间用逗号隔开即可。

**分组查询练习：**

# 查询男生与女生的人数

SELECT COUNT(\*),sex FROM students GROUP BY sex;

# 查询每个班级的学生人数

SELECT COUNT(\*) FROM students GROUP BY classid;

**思考：若查询每个班级的学生姓名会怎样？**

# 查询学生人数大于2的班级编号以及人数

SELECT classid,COUNT(\*) FROM students

GROUP BY classid HAVING COUNT(\*)>2;

**思考：若使用where筛选人数大于2会怎样？**

**having 与where 的区别：（面试）**

**语法不同：**

where 跟在 group by命令前面，是分组之前进行过滤，不满足where条件，不参与分组；

having 跟在group by命令后面，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过滤，而having可以。

**了解where与having 的根本区别：**

where子句是在group by分组和聚合函数之**前**，对表中的原始数据进行过滤；

having子句是在group by分组和聚合函数之**后**，对统计后的数据行进行过滤；

由于在执行where子句时，还没有计算聚合函数，所以where子句中不能使用聚合函数。

从性能的角度来说，having 子句中如果使用了分组字段作为过滤条件，应该替换成where子句；因为where可以在执行分组操作和计算聚合函数之前，过滤掉不需要的数据，性能会更好。

但未分组的字段不能使用having筛选，因为分组之后，未分组的字段已经不在查询范围了。

##### 1.5 排序查询

order by 是将其字段内的值进行升序或降序排序；以达到方便查看数据的目的。

排序查询语法：（order by）

select 字段列表 from 表名 order by 字段1 排序方式，字段2 排序方式，...；

**注意事项：**

order by 后可以跟需要排序的字段、函数、别名；

排序方式：**升序** ASC（默认）；**降序** DESC；

排序的方式应紧跟在字段后面，用空格隔开即可；

也可以跟多个排序的字段，多个字段之间用逗号隔开；

如果是多个字段排序，当第一个字段数值相同时，才会根据第二个字段值进行排序（不能随意调换顺序），以此类推。

如：字段1 asc,字段2 desc,......

**排序查询练习：**

# 查询学生的年龄，要求：按照年龄升序/降序排序

SELECT age FROM students ORDER BY age ASC;

SELECT age FROM students ORDER BY age DESC;

# 查询学生的年龄和地址编号，

# 要求：按照年龄升序，地址编号降序排序。

SELECT age,locationid FROM students

ORDER BY age,locationid DESC;

##### 1.6 分页查询

limit用于强制select语句返回指定的记录数。分页查询语法：（limit）

select 字段列表 from 表名 limit [offset，] size;

**注意事项：**

接受一个或两个参数，但参数必须是一个整数常量。

若果给定一个参数，则表示返回记录行的条数。

如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的条数。

**offset：**表示指定第一个返回记录行的偏移量（即从哪一行开始返回）；起始值为 0 。

**size：** 表示返回具体行数（查询行数）。

示例：

limit后跟 1 个值：limit 3 等同于 limit 0,3

表示查询前3条的数据。

limit后跟 2 个值：条数VS页数

条数：limit 4，6

表示查询第5到第10条的数据。

**计算方式：前：小值-1，后：大值-小值+1。**

页数（每页10条）：limit 20，10

表示查询第3页的数据。

**计算方式：前：(页数-1)×每页条数，后：每页条数。**

**分页查询练习：**

# 查询年龄最小/大的学生信息

SELECT \* FROM students ORDER BY age ASC LIMIT 1;

SELECT \* FROM students ORDER BY age ASC LIMIT 0,1;

# 查询地址编号最大的学生信息

SELECT \* FROM students ORDER BY locationid DESC LIMIT 1;

SELECT \* FROM students ORDER BY locationid DESC LIMIT 0,1;

# 查询第二到第四位的学生信息

SELECT \* FROM students LIMIT 1,3;

# 每页显示2个人，查询第2页的学生信息

SELECT \* FROM students LIMIT 2,2;

#### 2、多表查询

当查询结果的列来源于多张表时，需要将多张表连接成一个大的数据集，再选择合适的列返回。

SQL 99 语法多表连接查询类型：

内连接：关键字：[inner] join

等值连接，自连接，非等值连接。

外连接：关键字：left /right/full [outer] join

左外连接，右外连接，全外连接。

交叉连接：关键字：cross join

注意：MySQL5.7版本不支持全外连接。

##### 2.1 内连接

内连接返回连接表中符合连接条件和查询条件的数据行。

所谓的连接表就是数据库在做查询形成的中间表。

查询的结果为多个表匹配到的数据。

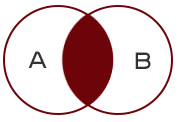
**2.1.1等值连接**

语法格式：

select \*

from 表A [inner] join 表B

on 表A.列=表B.列;



**注意事项：**

查询结果是两个表的交集部分；

inner可省略，默认为内连接；

表1，表2的顺序任意调换；

on后紧跟的连表条件；

支持给表取别名，但在此条SQL语句中使用表名的地方要使用别名,同时表名失效；

如果出现了多个表中列名一致的情况下， 需要用表名区别；

多个表连表时：表A join表B on表A.列=表B .列

join表C on表C.列=表B .列 ……依次类推。

**等值接练习：**

# 查询学生姓名

SELECT NAME FROM students;

# 查询班级名称

SELECT NAME FROM class;

# 查询学生姓名和班级名称

SELECT students.name,class.name

FROM students INNER JOIN class

ON students.classid=class.id;

# 查询学生姓名、班级名称、地址名称

SELECT stu.name,cla.name,loc.name

FROM students AS stu JOIN class AS cla

ON stu.classid=cla.id

JOIN locations AS loc

ON loc.id=stu.locationid;

**12.1.2非等值链接**

查询的结果为表1里某字段里的数据在表2里对应字段的等级范围。

语法格式:

select \*

from 表A [inner] join表B

On 表A.列 bewteen 表B.列 and 表B.列;

**非等值链接练习：**

# 查询学生的年龄以及年龄称呼

SELECT age,level.name

FROM students JOIN level

ON age BETWEEN low\_age AND high\_age;

**2.1.3自连接**

把一个表当两个表来使用。

注意：必须起别名，否则无法区分。

语法格式:

select \*

from 表1 AS 别名1 [inner] join表1 AS 别名2

On别名1 .列 = 别名2.列;

**自连接练习：**

# 查询地址名称以及上级地址名称

SELECT a.name,b.name

FROM locations AS a JOIN locations AS b

ON a.upid=b.id;

连表条件：地址表里的上级id=上级表里的id。

##### 2.2 外连接

外连接不但返回符合连接和查询条件的数据行，

还返回不符合连接条件，但符合查询条件的数据行。

**外连接分三类：**

左外连接 ：LEFT OUTER JOIN

右外连接 ：RIGHT OUTER JOIN

全外连接 ：FULL OUTER JOIN

**思考：为什么分左右？**

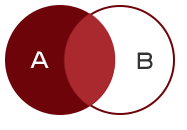
**2.2.1左外连接**

语法格式：

select \*

from 表A left [outer] join 表B

on 表A.列=表B.列;



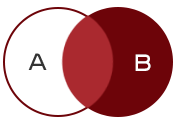
**2.2.2右外连接**

语法格式：

select \*

from 表A right [outer] join 表B

on 表A.列=表B.列;



**注意事项：**

**结果集**：主表的全部内容+从表中匹配到的内容+从表中未匹配到的内容以null补全。

外连接中表的**顺序**是**固定**的，不可以随意调换。

**左外**：left outer join **左边是主表，右边是从表；**

**右外**：right outer join **右边是主表，左边是从表**。

外连接时，join前的 outer可以省略不写，但是left或right

是不可以省略的，否则就不是外连接了。

**判断主从表**：通常否定哪个表，哪个表是从表；

如：没有学生的班级名称。班级表是主表，学生表是从表。

**筛选条件**：where筛选条件通常为从表的(主键)值为空；

筛选条件建议使用**主键**，因为主键约束的字段值是**非空且唯一**的。

如：没有学生的班级名称，筛选条件就是学生表的主键值为空：where students.id is null。

**外连接练习：**

# 查询班级名称和对应的学生姓名(内连接)

SELECT class.name,students.name

FROM class INNER JOIN students

ON students.classid=class.id;

# 查询所有班级名称和对应的学生姓名(左外连接)

SELECT class.name,students.name

FROM class LEFT OUTER JOIN students

ON students.classid=class.id;

# 查询没有学生的班级名称(左外连接)

SELECT class.name

FROM class LEFT OUTER JOIN students

ON students.classid=class.id

WHERE students.id IS NULL;

# 查询所有班级名称和对应的学生姓名(右外连接)

SELECT class.name,students.name

FROM students RIGHT OUTER JOIN class

ON students.classid=class.id;

# 查询没有学生的班级名称(右外连接)

SELECT class.name

FROM students RIGHT OUTER JOIN class

ON students.classid=class.id

WHERE students.id IS NULL;

**左外连接与右外连接的区别？**

**二者查询的结果集都是：**

主表的全部内容；从表里匹配到的内容，如果匹配不到的内容则以null值补全。

**语法不同：**

左外连接时，语法是 left [outer] join ；

左外连接时，语法是 right [outer] join 。

**主从表位置不同：**

左外连接时，主表在左边，从表在右边；

右外连接时，主表在右边，从表在左边。

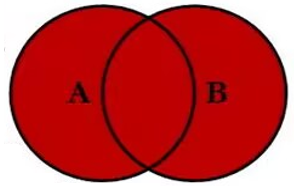
**12.2.3全外连接**

语法格式：

select \*

from 表A full [outer] join 表B

on 表A.列=表B.列；



**注意事项：**

**结果集**：A与B的交集部分+A中有但是B中没有的以NULL补全+B中有但是A中没有的以NULL补全；

由于MySQL5.7版本不支持全外连接语法，在此我们使用关联查询模拟全外连接结果。

**语法格式：**

select \* from 表A left [outer] join 表B on 表A.列=表B.列

**union**

select \* from 表A right [outer] join 表B on 表A.列=表B.列;

注意：执行是2条SQL一起执行。

A表 与 B表 的位置固定不变；

先用 左外连接 （A表主表，B表从表）匹配一条sql；

用 union 把两个表连起来；

再用 右外连接 （A表从表，B表主表）匹配一条sql；

最后把两条sql一起执行，显示结果与全外链接结果一样。

**全外连接练习：**

# 查询所有学生信息和所有班级信息（全外连接）

SELECT \*

FROM students FULL OUTER JOIN class

ON students.classid=class.id;

# 注意：MySQL 5.7版本不支持此语法。

SELECT \*

FROM students LEFT OUTER JOIN class

ON students.classid=class.id

**UNION**

SELECT \*

FROM students RIGHT OUTER JOIN class

ON students.classid=class.id;

##### 2.3 交叉连接

结果是笛卡尔积（缺少连表条件导致）。

**语法格式:**

select \*

from 表1 cross join表2；

# 使用交叉连接查询学生姓名和班级名称

SELECT students.NAME,class.NAME

FROM students CROSS JOIN class;

**分析使用场景：**

1、如果查询结果是两个表的交集部分，等值连接；

2、如果查询结果是一个表A的数据+另外一个表B的匹配的数据，使用左外右外连接都可；

3、如果查询结果是一个表A中有但是另外一个表B中没有的数据（不完全匹配），使用左外右外连接都可；

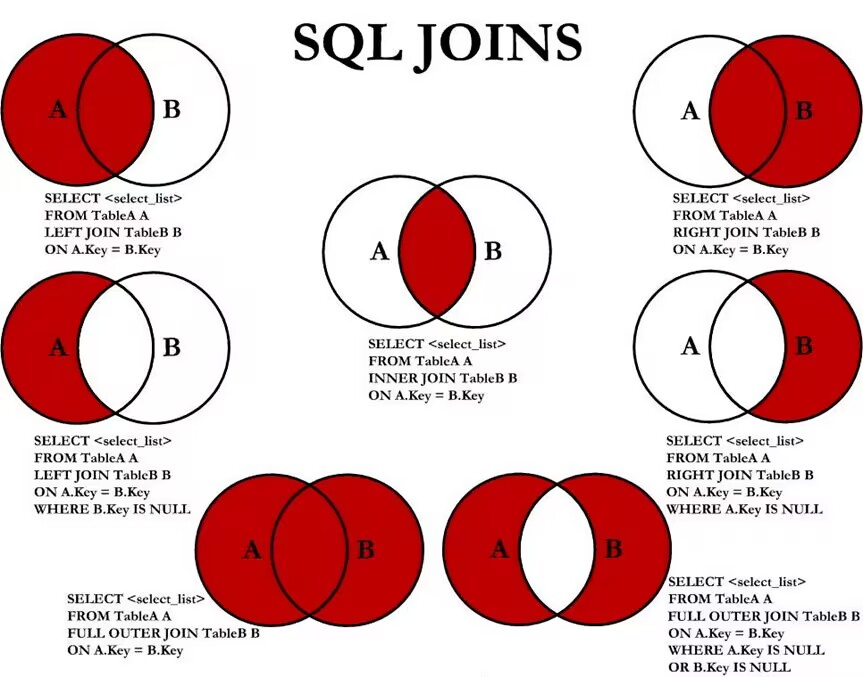
4、如果查询结果是A表与B表的交集部分+A表有B表没有的数据+

A表没有B表有的数据，使用全外连接；

**判断内连/外连技巧：通常肯定句式用内连，否定句式用外连；**

**判断主/从表技巧：否定词后所在的表就是从表；**

**判断筛选条件技巧：通常以从表的主键字段作为筛选字段。**



**SQL查询的基本原理：**

**单表查询：**根据WHERE条件过滤表中的记录，形成中间表（这个中间表对用户是不可见的）；然后根据SELECT的选择列选择相应的列进行返回最终结果。

**两表连接查询：**对两表求积（笛卡尔积）并用ON条件和连接连接类型进行过滤形成中间表；然后根据WHERE条件过滤中间表的记录，并根据SELECT指定的列返回查询结果。

**多表连接查询：**先对第一个和第二个表按照两表连接做查询，然后用查询结果和第三个表做连接查询，以此类推，直到所有的表都连接上为止，最终形成一个中间的结果表，然后根据WHERE条件过滤中间表的记录，并根据SELECT指定的列返回查询结果。

**SQL语句书写以及逻辑执行顺序：**



从这个顺序中我们不难发现，所有的查询语句都是从from开始执行的，在执行过程中，每个步骤都会为下一个步骤生成一个虚拟表，而生成个虚拟表将作为下一个执行步骤的输入。

**第一步：**首先对from子句中的前两个表执行一个笛卡尔乘积,此时生成虚拟表vt1 (选择相对小的表做基础表)。

**第二步：**接下来便是应用on筛选器, on中的逻辑表达式将应用到vt1中的各个行,筛选出满足on逻辑表达式的行,生成虚拟表vt2。

**第三步：**如果是outer join那么这一步就将添加外部行；left outer jion就把左表在第二步中过滤的添加进来 ,如果是right outer join那么就将右表在第二步中过滤掉的行添加进来,这样生成虚拟表vt3。

**第四步：**如果from子句中的表数目多余两个表,那么就将vt3和第三个表连接从而计算笛卡尔乘积,生成虚拟表,该过程就是一个重复1-3的步骤,最终得到一个新的虚拟表vt3。

**第五步：**应用where筛选器,对上一步生产的虚拟表引用where筛选器,生成虚拟表vt4。

注意：where与on的区别:先执行on ,后执行where ; on是建立关联关系在生成临时表时候执行, where是在临时表生成后对数据进行筛选的。

**第六步：**group by子句将中的唯一的值组合成为一组,得到虚拟表vt5。如果应用了group by ,那么后面的所有步骤都只能得到的vt5的列或者是聚合函数( count、sum、avg等 )。原因在于最终的结果集中只为每个组包含一行。

**第七步：**应用avg或者sum选项,为vt5生成vt6。

**第八步：**应用having筛选器,生成vt7。having筛选器是第一个也是为唯一个应用到已分组数据的筛选器。

**第九步：**处理select子句。将vt7中的在select中出现的列筛选出来生成vt8。

**第十步：**应用distinct子句,对vt8进行去重生成vt9。

**第十一步：**应用order by子句。按照排序方式对vt9进行排序,此时返回的是一个游标,而不是虚拟表。

**第十二步：**应用limit选项。生成vt10返回结果给请求者即用户。

#### 3、子查询

在一个 SQL 语句中,嵌入了另外一个 select 语句, 那么被嵌入的 select 语句称之为子查询语句。

**主查询：**

主要查询的对象,第一条 select 语句。

**主查询和子查询的关系：**

子查询是嵌入到主查询中的；

子查询是辅助主查询的，要么充当条件,要么充当数据源；

子查询是可以独立存在的语句，是一条完整的 select 语句。

注意:子查询可以嵌套多层，但每层需要用“()”括起来。

**子查询分类：**

标量子查询：一行一列

子查询返回的结果是一个数据。

列子查询：一列多行

子查询返回的结果是一列数据。

行子查询：一行多列

子查询返回的结果是一行数据。

表级子查询：多行多列

子查询返回的结果是多行多列数据。

**子查询书写位置：**

子查询书写的位置不同，充当的角色也不同。

**子查询写在 select 后面：**

把子查询的结果当成查询内容使用，只能跟标量子查询；

**子查询写在 from 后面：**

把子查询的结果当成数据源使用，必须给子查询起别名；

如果子查询的字段有函数作用时，必须给该字段起别名。

**子查询写在 where / having 后面：**

把子查询的结果当成筛选条件使用。

##### 3.1 标量子查询

常用**比较运算符**。

格式：主查询 where 条件 =(标量子查询)

**标量子查询练习：**

# 查询班级编号为1的学生信息

SELECT \* FROM students WHERE classid=1;

# 查询与王小明同班的学生信息

SELECT \* FROM students WHERE classid=(

SELECT classid FROM students WHERE NAME='王小明');

# 查询MySQL班的学生信息

# 连表查询

SELECT \* FROM class JOIN students ON class.id=students.classid WHERE class.NAME='MySQL班';

# 子查询

SELECT \* FROM students WHERE classid=(

SELECT id FROM class WHERE NAME='MySQL班');

# 查询女生的年龄以及男生的平均年龄

SELECT age,(SELECT AVG(age) FROM students

WHERE sex='男') AS 男生的平均年龄

FROM students WHERE sex='女';

# 进阶：查询班级人数比1号学生所在班级人数多的班级编号与人数

SELECT classid,COUNT(\*)

FROM students

GROUP BY classid

HAVING COUNT(\*)>(

SELECT COUNT(\*) FROM students WHERE classid=(

SELECT classid FROM students WHERE id=1));

##### 3.2 列子查询

**常用：in，any，all**

in 在...里面。

格式：主查询 where 条件 in (列子查询)

any 任意一个值

格式：主查询 where 列 = any (列子查询)

表示等于(列子查询)中任意一个值即可；等价于 in。

all 所有值

格式：主查询 where 列 != all(列子查询)

表示不等于(列子查询)中所有值；等价于 not in。

**注意：any和all只能用于子查询里面，不能用于普通查询。**

**列子查询练习：**

# 查询与学生编号为1、3班级相同的学生姓名（in）

SELECT NAME FROM students WHERE classid IN(

SELECT classid FROM students WHERE id IN(1,3));

# 查询年龄比1班任意年龄小的学生姓名、年龄、班级编号（any）

SELECT NAME,age,classid FROM students WHERE age<ANY(

SELECT age FROM students WHERE classid=1)

AND classid!=1;

# 查询年龄比1班所有年龄小的学生姓名和年龄（all）

SELECT NAME,age,classid FROM students WHERE age<ALL(

SELECT age FROM students WHERE classid=1);

# 查询学生编号为1、3的学生信息

SELECT \* FROM students WHERE id IN(1,3);

# any 与 all 不能用于普通查询。

SELECT \* FROM students WHERE id=ANY(1,3);

虽然=any等价于 in，但any 与 all后面的小括号中只能是SQL语句，不能是常量值。

##### 3.3 行子查询

筛选条件时，把多个结果一起筛选，不过要注意前后顺序不能颠倒。

格式：主查询 where (条件1,条件2) =(行子查询)

**行子查询练习：**

# 查询1班年龄最高的学生信息

SELECT \* FROM students WHERE (classid,age)=(

SELECT classid,MAX(age) FROM students WHERE classid=1);

##### 3.4 表子查询

通常放在from命令后面，把子查询的结果当做数据源使用。

格式：主查询from (表子查询) AS 表别名；

**注意事项：**

必须给表子查询起别名；

如果子查询的字段有函数作用时，必须给该字段起别名。

**表级子查询练习：**

# 把每个班的平均年龄当做数据源，

# 查询最低平均年龄。

SELECT MIN(avg\_age)

FROM (

SELECT AVG(age) AS avg\_age,classid

FROM students

GROUP BY classid) AS a;

**标量子查询：常用比较运算符。**

格式：主查询 where 条件 =(标量子查询)

**列子查询：常用in，any，all**

in 在...里面。

格式：主查询 where 条件 in (列子查询)

any 任意一个值

格式：主查询 where 列 = any (列子查询)

表示等于(列子查询)中任意一个值即可；等价于 in。

all 所有值

格式：主查询 where 列 != all(列子查询)

表示不等于(列子查询)中所有值；等价于 not in。

**注意：any和all只能用于子查询里面，不能用于普通查询。**

**标量子查询练习：**

查询与140号员工的部门编号

SELECT department\_id

FROM employees

WHERE employee\_id = 140;

查询与140号员工同部门的员工姓名

SELECT first\_name,last\_name

FROM employees

WHERE department\_id = (

SELECT department\_id

FROM employees

WHERE employee\_id = 140);

查询部门人数比202号员工所在部门人数多的部门编号与人数

SELECT department\_id, COUNT(employee\_id) 人数

FROM employees

GROUP BY department\_id

HAVING人数 >(

SELECT COUNT(\*)

FROM employees

WHERE department\_id=(

SELECT department\_id

FROM employees

WHERE employee\_id=202));

查询102员工的姓名，工资以及所在部门的平均工资

SELECT first\_name,last\_name,salary,

(SELECT AVG(salary)

FROM employees

WHERE department\_id=(

SELECT department\_id

FROM employees

WHERE employee\_id=102)) 平均工资

FROM employees

WHERE employee\_id=102;

**列子查询练习：**

查询与员工编号为105,109,114部门相同的员工名

SELECT first\_name

FROM employees

WHERE department\_id IN (

SELECT department\_id

FROM employees

WHERE employee\_id IN(105,109,114));

查询其他工种中比工种为'IT\_PROG'工种任一工资低的

员工姓名，工种，工资

SELECT first\_name,last\_name,job\_id,salary

FROM employees

WHERE salary < (

SELECT MAX(salary)

FROM employees

WHERE job\_id = 'IT\_PROG')

AND job\_id <> 'IT\_PROG';

SELECT first\_name,last\_name,job\_id,salary

FROM employees

WHERE salary < ANY(

SELECT salary

FROM employees

WHERE job\_id = 'IT\_PROG')

AND job\_id <> 'IT\_PROG';

查询其他工种中比工种为'IT\_PROG'工种所有工资低的

员工姓名，工种，工资

SELECT first\_name,last\_name,job\_id,salary

FROM employees

WHERE salary < (

SELECT MIN(salary)

FROM employees

WHERE job\_id = 'IT\_PROG')；

SELECT first\_name,last\_name,job\_id,salary

FROM employees

WHERE salary < ALL(

SELECT salary

FROM employees

WHERE job\_id = 'IT\_PROG')；

ANY和ALL只能用于子查询里面，不能用于普通查询。

查询部门编号为10，20，30的员工的姓名

SELECT first\_name,last\_name

FROM employees

WHERE department\_id IN (10,20,30);

**SELECT first\_name,last\_name**

**FROM employees**

**WHERE department\_id =ANY (10,20,30);**

**行子查询练习：**

查询50号部门中工资最高的员工信息

SELECT \*

FROM employees

WHERE (department\_id,salary)=(

SELECT department\_id,salary

FROM employees

WHERE department\_id=50

ORDER BY salary DESC

LIMIT 1);

**表级子查询练习：**

把每个部门的平均工资当成一张表，查询最低平均工资是多少？

SELECT MIN(m)

FROM (SELECT AVG(salary) m

FROM employees

GROUP BY department\_id) s;

注意：

把子查询的结果当成数据源使用，必须给子查询起别名；

如果子查询的字段有函数作用时，必须给该字段起别名。

### 七、DML数据操作语言

DML语言主要用来对表中的数据进行增加、修改、删除操作。

#### 1、添加数据

**1.1 添加一行数据**

**语法格式一：部分字段设置值。**

**insert into 表名(列1,列2) values(值1,值2);**

注意：值的顺序与书写字段的顺序对应。

**语法格式二：所有字段设置值。**

**insert into 表名 values(值1,值2,...);**

注意：表名后省略字段，默认全表添加，值的顺序与表中字段的顺序对应。8.0以后版本表名后必须写列名。

**添加一行数据练习：**

# 准备数据：仿照locations表创建一个locations\_copy表

CREATE TABLE locations\_copy LIKE locations;

# 往locations\_copy表添加一条数据：

# 地址编号 1 ，地址名称 中国。

INSERT INTO locations\_copy(id,NAME) VALUES(1,'中国');

# 往locations\_copy表添加一条数据：

# 地址编号 2 ，地址名称 河南，上级编号 1；

INSERT INTO locations\_copy VALUES(2,'河南',1);

**1.2 添加多行数据**

**语法格式一：所有字段设置值。**

**insert into 表名**

**values(值1,值2,...)，(值1,值2,...)，...;**

**语法格式二：部分字段设置值。**

**insert into 表名(列1,列2)**

**values(值1,值2)，(值1,值2)，...;**

注意：

每行值必须用小括号括起来；

小括号与小括号之间用逗号分开；

最后一行数据用分号结尾。

**添加多行数据练习：**

# 往locations\_copy表添加三条数据：

# 地址编号 3 ，地址名称 郑州，上级编号 2；

# 地址编号 4 ，地址名称 周口，上级编号 2；

# 地址编号 5 ，地址名称 淮阳，上级编号 4；

INSERT INTO locations\_copy

VALUES(3,'郑州',2),(4,'周口',2),(5,'淮阳',4);

# 准备数据：仿照class表创建一个class\_copy表

CREATE TABLE class\_copy LIKE class;

# 往class\_copy表添加三条数据：

# 班级名称 MySQL班，Linux班，Python班。

INSERT INTO class\_copy(NAME)

VALUES('MySQL班'),('Linux班'),('Python班');

**# 添加失败的原因是主键未设置自增。**

#### 2、修改数据

**2.1 单表修改**

**语法格式：**

update 表 set 列1=值1,列2=值2,列3=值3,...

[where 筛选条件]

**注意事项：**

改数据时可以使用条件筛选，

不筛选表示修改指定列全部值；

修改多个列时，列名与列名之间用逗号分割。

**单表修改练习：**

# 建议：工作中需要改数据时，先把原始数据备份。

# 把students表数据备份到students\_copy表中。

CREATE TABLE students\_copy SELECT \* FROM students;

# 把王小明的手机号修改为13123456789

UPDATE students

SET mobile='13123456789'

WHERE NAME='王小明';

# 把王小明的年龄加1岁

UPDATE students

SET age=age+1

WHERE NAME='王小明';

**2.2 连表修改**

**内连接语法格式:**

update 表1 join 表2 on 表1.列=表2.列

set 列1=值1,列2=值2,列3=值3,...

[where 筛选条件]

**外连接语法格式:**

update表1 left/right join 表2 on 表1.列=表2.列

set 列1=值1,列2=值2,列3=值3,...

[where 筛选条件]

**注意事项：**

join后面要紧跟on连表条件(join与on之间不能有其他命令)

然后再跟set修改列=值，顺序不能颠倒。

**连表修改练习：**

# 把MySQL班的学生年龄改为22岁

UPDATE students INNER JOIN class

ON classid=class.id

SET age=22

WHERE class.name='MySQL班';

# 把没有学生的班级名称改为Java班

UPDATE class LEFT OUTER JOIN students

ON classid=class.id

SET class.name='Java班'

WHERE students.id IS NULL;

#### 3、删除数据

**3.1 单表删除**

**删除全表数据语法格式：**

delete from 表名；

truncate 表名；

**删除部分数据语法格式：**

delete from 表名 where 筛选条件；

**单表删除练习：**

# 删除王小明的信息

DELETE FROM students WHERE NAME='王小明';

# 删除学生表信息

DELETE FROM students;

# 删除class\_copy表信息

TRUNCATE class\_copy;

思考：delete 与 truncate 有什么区别呢？

**delete和truncate区别：**

delete：删除表中的内容，不删除表结构，但不释放空间。

truncate：删除表中的内容，不删除表结构，释放空间。

**语法不同：**

delete可以在后续加上where进行针对行的筛选删除；

truncate后面只能加上表名，直接删除全表数据，不能进行筛选。

**内存空间不同：**

使用delete命令删除数据后，重写数据，自增值会从删除前的最后一行续写；

使用truncate命令删除数据后，重写数据，会从1开始；

truncate比较节省空间。

**处理速度不同：**

truncate删除数据时是直接从1开始，即全部清空；

而delete删除数据时需要先得到行数，再按行删除；

所以truncate删除速度比delete更快。

**返回信息不同：**

使用delete命令删除数据时，返回受影响行数量；

使用truncate命令删除数据时，不返回受影响行数量。

**3.2 连表删除**

**内连接语法格式：**

delete 表1,表2

from 表1 join 表2 on 表1.列=表2.列

[where 筛选条件]；

**外连接语法格式：**

delete 表1,表2

from 表1 left/right join 表2 on 表1.列=表2.列

[where 筛选条件]；

**注意事项：**

如果给表起别名,那么在使用表名的位置必须只能使用别名,

delete后面也要使用表的别名；

如果delete后面只跟表1，表示只删除表1中的数据；

如果delete后面只跟表2，表示只删除表2中的数据；

如果delete后面跟表1，表2，表示删除表1，表2中的数据。

**连表删除练习：**

# 把class表中的数据备份到class\_copy表中

INSERT INTO class\_copy SELECT \* FROM class;

# 把students表中的数据备份到students\_copy表中

INSERT INTO students\_copy SELECT \* FROM students;

# 删除MySQL班的学生信息

DELETE students\_copy

FROM students\_copy JOIN class\_copy

ON classid=class\_copy.id

WHERE class\_copy.name='MySQL班';

# 删除王大锤以及他所在班级的信息

DELETE students\_copy,class\_copy

FROM students\_copy JOIN class\_copy

ON classid=class\_copy.id

WHERE students\_copy.name='王大锤';

# 删除没有学生的班级信息

DELETE class\_copy

FROM class\_copy LEFT JOIN students\_copy

ON class\_copy.id=classid

WHERE students\_copy.id IS NULL;

**注意：**若有外键作用时，必须先去除外键约束，才能删除数据。

因为父表的数据若被子表引用时，父表的数据不允许被修改或者删除。

**去除外键方式：**

鼠标右键子表--改变表--点2个索引--勾选目标--点击-号--点3个外部键--勾选目标--点击-号--保存。

### 八、数据类型

MySQL常用数据类型：**数值型、字符型、日期型**等。

#### 1、数值型

**整数：**

int 取值范围（-231-- 231-1）或者（0-- 232-1）

有符号范围（-2147483648 ~2147483647）。

无符号范围（0 ~ 4294967295）。

bigint 取值范围（-263 -- 263-1）或者（0—264-1）。

有符号范围（- 9223372036854775808 ~9223372036854775807）。

无符号范围（0 ~ 18446744073709551615）。

**小数：**

float(M,D) 有效位数是7位左右（精度低，一般不用）。

double(M,D) 有效位数是15位左右（常用）。

decimal(M,D) 有效位数是28位左右（金融类常用）。

**M** 表示数值的整个长度（整数长度+小数长度）；

**D** 表示小数点后的长度（小数点不计算长度）。

如：123.45 float（5,2）

表示共存5位数，小数占2位，整数占3位。

123.12345--123.123 转换过程：

123.12345先转换成1.2312345\*102

保留小数点后5-6位后转换成1.23123\*102也可以写成123.123

#### 2、字符型

varchar 储存可变长度字符串，范围（0~65533）。

char 储存固定长度字符串。

**注意：**

在char或varchar定义的字段中，

一个中文、字母、数字、符号都是占一个字符的长度。

如：A定义： char(5) ，B定义： varchar(5)

若要存储 'abc'字符串。

A储存的是5位：‘abc’+ 2位空格，查询时自动去空格；

B储存的是3位：字符串实际长度‘abc’。

**varchar和char的区别：**

char存储的是固定长度字符串；

varchar存储的是可变长度字符串；

所以varchar比char更节省空间。

char与varchar在进行数据库设计时，必须设置长度；

char不设置长度不会提示错误，但是默认长度不一定能满足需求；

varchar在进行数据库设计时，必须设置长度，否则会报错。

#### 3、日期型

date 只保存日期。

数据显示类型：YYYY-MM-DD （年月日）

time 只保存时间。

数据显示类型：HH:II:SS（时分秒）

year 只保存年。

数据显示类型：YYYY （年）

datetime 保存日期加时间。

数据显示类型：YYYY-MM-DD HH:II:SS （年月日 时分秒）

范围（1000-01-01 00:00:00 ~ 9999-12-31 23:59:59）

### 九、约束类型

约束用于限制数据输入或数据更新的规则。在创建表的时候,可以给表的字段添加相应的约束。

**约束的作用:**

确保在表中只能插入符合规则的数据，以确保数据的合法性、有效性、一致性和完整性。

防止数据库中存在不符合语义规定的数据；

防止错误信息的输入输出造成无效操作或错误信息。

**约束的类型:**

**主键**约束：**primary key**

用于唯一标识表中的每一行记录，确保数据唯一性和完整性

**外键**约束：**foreign key**

用于确保表之间数据的关系正确性，保证引用表（子表）中的数据必须存在于被引用的表（父表）中。

**唯一**约束：**unique**

用于确保列中的值不重复。

**非空**约束：**not null**

用于确保列中的值不为空。

默认约束：**default**

用于设置默认值，当插入数据时如果未提供该值，则落库默认值。

**检查**约束：**check**

用于确保列中的值符合特定的条件或表达式。MySQL5.7版本不支持。

**思考：主键约束、外键约束、唯一约束之间有什么区别？**

### 十、DDL定义语言

DDL语言主要用来定义数据库、表、列。

#### 1、库的查询、添加、修改、删除语法

**1.1查看库语法格式**

SHOW DATABASES;

表示显示当前数据库服务下的所有库名。

# 查看数据库服务

SHOW DATABASES;

**1.2创建数据库语法格式**

create database 库名;

**注意：**库名起名时尽量全字母（不要使用中文）。

# 创建一个数据库并命名为abc

CREATE DATABASE abc;

**1.3修改库**

**修改库名：**

在C盘下的隐藏文件ProgramData找到Mysql里面的Mysql Server 5.7里面的DATA（DATA里面存放了所有的表数据），找到对应的库，右键重命名就行；在sqlyog中刷新一下就ok了。

常见路径： C:\ProgramData\MySQL\MySQL Server 5.7\Data

**修改库字符集语法格式：**

alter database 库名 character set = '字符集';

**字符集：utf8，gbk ；**作用是为了防止中文出现乱码。

**注意：**

修改库字符集要在创建表之前修改，因为只对修改字符集后再创建的表起作用，修改字符集之前创建的表不起作用。

# 修改abc库字符集为utf8/gbk

ALTER DATABASE abc CHARACTER SET='utf8';

ALTER DATABASE abc CHARACTER SET=‘gbk';

**1.4删除数据库语法格式**

drop database 库名;

删除abc数据库

DROP DATABASE abc;

#### 2、表的查询、添加、修改、删除语法

**2.1查看表结构**

第一种：DESC 表名;

第二种：show columns from 表名;

# 查看学生表结构：

DESC students;

SHOW COLUMNS FROM students;

**2.2创建表**

**语法格式一：**

create table 新表名 like 旧表名;

**语法格式二：**

create table 表名 （列名1 数据类型 [comment '注释'],

列名2 数据类型 表级约束类型 [comment '注释'],......，

表级约束类型(列名)，

[constraint约束名] foreign key(子表列名)

references父表名(父表列名),......）;

**注意事项：**

起表名与列名时遵循‘见名知意’，英文+数字+下划线；

**语法一**创建表时，必须有旧表；

**语法二**创建表时，参考以下规则：

列名、数据类型、约束之间使用空格分隔；

列跟列之间使用逗号分隔；

最后一列不需要添加任何符号，结束的分号放在小括号外；

**列级**约束类型：主键，唯一，非空，默认；

**表级**约束类型：主键，唯一，外键；

constraint 作用是给约束起别名，可以省略，但省略后不能给约束起别名；

comment注释可写可不写，但是一般建议写上，更加明确。

**创建表练习：**

仿照class表创建一个class\_copy2表。

CREATE TABLE class\_copy2 LIKE class;

创建schools库，

库中有：students、class、locations 三个表。

students表：

id 学生编号 主键

name 学生姓名 非空

sex 性别 默认值‘男’

age 年龄 检查‘0--150岁’

birthdate 出生日期

mobile 手机号 唯一

classid 班级编号 外键

locationid 地址编号 外键

class表：

id 班级编号 主键

name 班级名称 非空

locations表：

id 地址编号 主键

name 地址名称 非空

upid 上级编号

**思路：**

**第一步：创建schools库**

**第二步：创建class表（父表）**

**第三步：创建locations表（父表）**

**第四步：创建students表（子表）**

# 1.创建schools库

CREATE DATABASE schools;

# 2.创建class表（父表）

CREATE TABLE class(

id INT PRIMARY KEY COMMENT'班级编号',

NAME VARCHAR(20) NOT NULL COMMENT'班级名称');

# 3.创建locations表（父表）

CREATE TABLE locations(

id INT COMMENT'地址编号',

NAME VARCHAR(20) NOT NULL COMMENT'地址名称',

upid INT COMMENT'上级编号',

PRIMARY KEY(id));

# 4.创建students表（子表）

CREATE TABLE students(

id INT COMMENT'学生编号',

NAME VARCHAR(30) NOT NULL COMMENT'学生姓名',

sex CHAR(1) DEFAULT'男' COMMENT'性别',

age INT COMMENT'年龄',

birthdate DATE COMMENT'出生日期',

mobile CHAR(11) COMMENT'手机号',

classid INT COMMENT'班级编号',

locationid INT COMMENT'地址编号',

PRIMARY KEY(id),UNIQUE(mobile),

FOREIGN KEY(classid) REFERENCES class(id),

CONSTRAINT waijian FOREIGN KEY(locationid) REFERENCES locations(id));

**了解备份表：**

备份表时有以下情形：

备份全部表结构。

备份部分表结构。

备份全部表结构与全部数据。

备份全部表结构与部分数据。

备份部分表结构与全部数据。

备份部分表结构与部分数据。

**表结构：**字段名以及数据类型，注释，约束，索引等。

**数据：**表内所存储的数据。

**备份全部表结构语法格式：**

create table 表2 like 表1;

# 备份students表全部结构到students\_copy2表中。

CREATE TABLE students\_copy2 LIKE students;

**备份部分结构语法格式：**

create table 表2

select [表1]列1, [表1]列2... from 表1 where 筛选条件不成立;

注意：where条件必须不能成立。

如：0=1或者1=2之类的。

# 备份students表部分表结构到students\_copy3表中。

CREATE TABLE students\_copy3

SELECT NAME,mobile FROM students WHERE 1=2;

**备份全部表结构与全部数据语法格式：**

create table 表2 select \* from 表1;

# 备份students表全部结构和全部数据到students\_copy4表中。

CREATE TABLE students\_copy4

SELECT \* FROM students;

**备份全部表结构与部分数据语法格式：**

create table 表2

select \* from 表1 where 筛选条件;

# 备份students表全部结构和部分数据到students\_copy5表中。

CREATE TABLE students\_copy5

SELECT \* FROM students WHERE id=1;

**备份部分表结构与全部数据语法格式：**

create table 表2

select [表1]列1,[表1]列2... from 表1；

# 备份students表部分表结构与全部数据到students\_copy6表中。

CREATE TABLE students\_copy6

SELECT NAME,mobile FROM students;

**备份部分表结构与部分数据语法格式：**

create table 表2

select [表1]列1,[表1]列2... from 表1 where 筛选条件成立；

# 备份students表部分结构与部分数据到students\_copy7表中。

CREATE TABLE students\_copy7

SELECT NAME,mobile FROM students WHERE id IN(1,2);

**2.3修改表**

修改表的常见操作：

rename 修改表名。

change 修改表内字段名。

modify 修改表内字段的数据类型、约束、注释。

add 给表添加字段。

drop 删除表内字段。

**2.3.1 修改表名语法**

alter table 表名

rename to 新表名;

# 将students\_copy表名修改为students\_copy1

ALTER TABLE students\_copy

RENAME TO students\_copy1;

**2.3.2 修改表内字段名语法**

alter table表名

change old列名 new列名 数据类型 [注释']；

**注意：**不加注释会把原有的注释修改。

# 把students\_copy1表name修改为sname

ALTER TABLE students\_copy1

CHANGE NAME sname VARCHAR(30) NOT NULL COMMENT'姓名';

**2.3.3 修改表内字段的数据类型与约束语法**

alter table表名

modify column 列名 数据类型 约束类型 [注释]；

# 修改students\_copy1表sname数据类型、约束类型、注释

ALTER TABLE students\_copy1

MODIFY sname CHAR(20) UNIQUE COMMENT'学生名';

**3.3.4 给表添加字段语法**

alter table表名

add 列名 数据类型 约束类型 [注释]；

# 给students\_copy1表添加height字段

ALTER TABLE students\_copy1

ADD height DOUBLE(3,2) NOT NULL COMMENT'身高';

**3.3.5 删除表内字段语法**

alter table表名

drop 列名；

注意：删除字段时，改字段内的数据也会删除。

# 删除students\_copy1表height字段

ALTER TABLE students\_copy1

DROP height;

**2.4删除表**

drop table 表名；（慎用！！！）

# 删除class\_copy表

DROP TABLE class\_copy;

### 十一、视图

视图本质就是对查询结果的封装。

若一个复杂的查询，在多个地方被使用；一旦需求发生了改变，需要更改SQL语句，则需要在多个地方进行修改，维护起来非常麻烦。

**解决方案：定义视图。**

**视图的作用：安全、效率。**

**数据更加安全：**

将可以开放的数据创建为视图，给访问者访问视图的权限，将需要隐藏的数据不对外提供。

**提高效率：**

将复杂的sql创建为视图，下次再需要执行该sql时，可以直接访问视图。

**创建视图语法格式：**

create view 视图名称 as select 语句;

# 将学生表的姓名、班级编号封装为视图

CREATE VIEW v\_stu AS

SELECT NAME,classid FROM students;

# 将学生的姓名、班级名称封装为视图

CREATE VIEW v\_stu\_cla AS

SELECT s.name AS 学生姓名,c.name AS 班级名称

FROM v\_stu s JOIN class c

ON classid=c.id;

注意：封装的视图中字段名不能重复。

**使用视图语法格式：**

SELECT \* FROM 视图名;

此时把视图当做数据源使用。

# 查看v\_stu视图信息

SELECT \* FROM v\_stu;

**删除视图语法格式：**

drop view 视图名称;

# 删除v\_stu视图

drop view v\_stu;

### 十二、存储过程

存储过程是一次编译，就会被缓存起来，下次使用就直接命中缓存中已经编译好的 SQL 语句即可，不需要重复编译。

减少网络交互，减少网络访问流量；从而达到减少传输时间、提高性能的目的。

**创建存储过程语法格式：**

DELIMITER//

CREATE PROCEDURE 存储过程名(参数列表)

BEGIN

SQL语句;

END //

**delimiter** /dɪˈlɪmɪtər/定界符。

如：DELIMITER// 表示将//定义为SQL语句的结束符。

# 将学生的姓名、班级名称、地址名称创建为存储过程

DELIMITER $$

CREATE PROCEDURE stu\_cla\_loc()

BEGIN

SELECT students.name,class.name,locations.name

FROM students JOIN class

ON students.classid=class.id

JOIN locations

ON locations.`id`=students.locationid;

END $$

DELIMITER;

**调用存储过程语法格式：**

CALL 存储过程名（参数列表）;

CALL stu\_cla\_loc();

**删除存储过程语法：**

DROP PROCEDURE存储过程名;

DROP PROCEDURE stu\_cla\_loc;

### 十三、函数

函数是一种功能（算法/方法）。

**通常我们使用的函数有两种：**

MySQL内置函数，自定义函数。

**内置函数：**MySQL已经封装好的函数，直接调用即可。

**自定义函数：**一次编译，就会被缓存起来，下次使用就直接调用已经编译好的 sql 即可，不需要重复编译大量代码；从而达到提高效率的目的。

#### 1、自定义函数

**自定义函数语法格式：**

DELIMITER //

CREATE FUNCTION 函数名(参数) RETURNS 数据类型

BEGIN

SQL语句;

END//

创建一个求和的函数

DELIMITER//

CREATE FUNCTION he(a INT,b INT) RETURNS INT

BEGIN

RETURN a+b;

END//

DELIMITER;

**调用函数语法格式：**

SELECT 函数名(参数);

SELECT he(1,2);

**删除函数语法格式：**

DROP FUNCTION 函数名;

DROP FUNCTION he;

#### 2、内置函数

MySQL数据库内置的函数很丰富，我们仅了解一部分即可。

**字符串函数：**处理字符串的函数；

**数学函数：**处理数字的函数；

**日期和时间函数：**处理日期和时间的函数；

**聚合函数：**处理一组数据的函数。

**2.1字符串函数**

concat(str1,str2,...) 将多个字符串拼接成一个字符串。

SELECT CONCAT(12,34,'ab');

upper(str)将字符串中的所有字符转换成大写。

lower(str)将字符串中的所有字符转换成小写。

SELECT UPPER('abcd');

SELECT LOWER('ASDF');

length(str) 返回字符串的长度。

SELECT LENGTH('我要好好学习mysql');

left(str,len)返回字符串str的左端len个字符；

right(str,len)返回字符串str的右端len个字符；

substring(str,pos,len)返回字符串str的位置pos起len个字符。

注意：substring不写len长度，默认截到末尾位置。

SELECT LEFT('我要好好学习mysql',1);

SELECT RIGHT('我要好好学习mysql',5);

SELECT SUBSTRING('我要好好学习mysql',7);

SELECT SUBSTRING('我要好好学习mysql',3,4);

trim(str)返回删除了左右两边空格的字符串str；

ltrim(str)返回删除了左边空格的字符串str；

rtrim(str)返回删除了右边空格的字符串str。

SELECT TRIM(' 好好学习 ');

SELECT LTRIM(' 好好学习 ');

SELECT RTRIM(' 好好学习 ');

replace(str1,str2,str3) 使用str3替换str1中的str2。

注意：若str2不存在str1中，则str3替换不成功。

SELECT REPLACE(‘好好学习’,‘好好’,‘天天');

SELECT REPLACE(‘好好学习’,‘努力’,‘天天');

lpad(str,len,padstr)向左填充。

rpad(str,len,padstr)向右填充。

注意：len表示总长度，padstr表示填充的字符串

SELECT LPAD('abc',6,'\*');

SELECT RPAD('abc',6,'\*');

**2.2数学函数**

round(n,d) 将n进行四舍五入。

n表示原数，d 表示小数位数。

注意：若省略d，表示默认四舍五入到整数位。

SELECT ROUND(1.68);

SELECT ROUND(1.68,1);

ceil(n)向上取整。

floor(n)向下取整。

SELECT CEIL(1.68);

SELECT FLOOR(1.68);

rand() 返回0-1之间的随机数（返回的值包含0，不包含1）。

注意：（）里面不填写参数。

获取一个0--1之间的随机数

SELECT RAND();

获取一个0--100之间的随机整数

SELECT CEIL(RAND()\*100);

获取100--200之间的随机数整数

SELECT FLOOR(100+RAND()\*101);

PI() 返回圆周率。

注意：小括号内无参数。

SELECT PI();

ABS(X) 返回X的绝对值。

SELECT ABS(-6);

SELECT ABS(6);

POW(X,Y) 返回X的Y次幂。

SELECT POW(2,3);

SQRT(X) 返回X的平方根。

SELECT SQRT(9);

**2.3日期和时间函数**

now() 与sysdate() 函数显示系统年月日时分秒。

SELECT NOW();

SELECT SYSDATE();

curdate() 获取系统年月日。

SELECT CURDATE();

curtime() 获取系统时分秒。

SELECT CURTIME();

year(date)返回日期的年。

SELECT YEAR(SYSDATE());

month(date)返回日期的月。

SELECT MONTH(SYSDATE());

day(date)返回日期的日。

SELECT DAY(SYSDATE());

**日期格式与字符串之间互转**

str\_to\_date(str,format)字符串转日期格式。

表示将字符串通过format格式转换为日期格式。

str表示需要转化的字符串；

fromat表示转化成的格式。

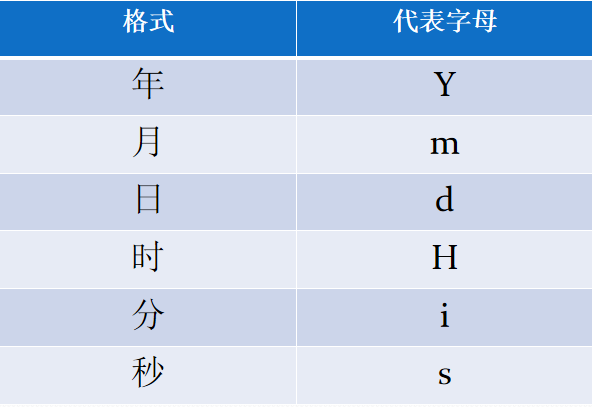
date\_format(date,format)日期格式转字符串。

表示将日期格式按照format格式转换为字符串。

date表示需要转化的字符串；

fromat表示格式。

**format格式：年月日时分秒**



**注意：** 大写Y表示完整年份，小写y表示缩写年份。

大写H表示24小时，小写h表示12小时。

**注意事项：**

**字符串转日期格式**时：str\_to\_date(str,format)

format格式要求与字符串的年月日时分秒**一一对应**。

如:字符串为年月日时分秒, format格式顺序为YmdHis。

示例：

字符串：'2021/12/13' format格式顺序：'%Y/%m/%d'

字符串：'9/26/2020 ' format格式顺序：'%m/%d/%Y'

字符串：'2021-11-12 13:14:15'

format格式顺序：'%Y-%m-%d %H:%i%s'

**日期格式转字符串**时：date\_format(date,format)

转换时，是按照format格式显示字符串；

format格式不要求与日期格式的年月日时分秒一一对应。

示例：

日期格式：2021-12-13

format格式可以写成：

'%Y-%m-%d' 或 '%Y-%m-%d'

'%Y年%m月%d日' 或 '%m/%d/%Y'

**理解：**

任何字符串写法转日期格式，最终显示日期格式。

日期格式转字符串，可以根据format格式任意写。

将字符串‘1988年1月3日’转化成日期格式

SELECT STR\_TO\_DATE('1988年1月3日','%Y年%m月%d日');

SELECT STR\_TO\_DATE('1988/1/3','%Y/%m/%d');

SELECT STR\_TO\_DATE('1988-1-3','%Y-%m-%d');

SELECT STR\_TO\_DATE('11/13/1988','%m/%d/%Y');

将字符串 ‘2021-12-13 14:15:16’ 转换成日期格式

SELECT DATE\_FORMAT('2020-12-12 14:13:19',

'%Y-%m-%d- %H:%i:%s');

将系统时间转换成字符串 （年月日时分秒）

SELECT DATE\_FORMAT(NOW(),'%Y-%m-%d );

SELECT DATE\_FORMAT(NOW(),'%Y-%m-%d %H:%i:%s');

SELECT DATE\_FORMAT(NOW(),'%Y/%m/%d %H:%i:%s');

SELECT DATE\_FORMAT(NOW(),'%Y年%m月%d日 %H时%i分%s秒');

SELECT DATE\_FORMAT(NOW(),'%m/%d/%Y %H:%i:%s');

# 查询出生日期为2001年08月15日的学生信息。

# 1.日期类型转换为字符串类型

SELECT \*

FROM students

WHERE DATE\_FORMAT(birthdate,'%Y年%m月%d日')='2001年08月15日';

# 2.字符串类型转换为日期类型

SELECT \*

FROM students

WHERE birthdate=STR\_TO\_DATE('2001年08月15日','%Y年%m月%d日');

**时间戳**

时间戳是把时间转换成从 1970-01-01 00:00 开始到指定时间所经过的秒数,这个秒数值就是指定时间对应的时间戳。

默认时间戳类型的取值范围为：1970-01-01 00:00:01 至2038-01-19 03:14:07。

数据库存储时间时：先将指定时间转换时间戳，再将时间戳的数值落库到数据库中。

数据库读取时间时：先将时间戳的数值转换为时间，再按照要求返回给客户端。

**设计表时设置时间戳字段存储数据，是因为时间戳是数值，而数值方便比较和运算。**

**日期与时间戳的相互转换:**

unix\_timestamp(datetime)日期转换为时间戳。

表示将日期转换为时间戳的数值。

datetime表示日期。

from\_unixtime(unix\_timestamp,format)时间戳转换为日期格式。

表示将时间戳的数值按照format格式转换为日期。

unix\_timestamp 表示时间戳；

format 表示格式。

将日期2020-10-11 12:13:14转换为时间戳(秒值)

SELECT UNIX\_TIMESTAMP('2020-10-11 12:13:14');

将现在转换为时间戳(秒值)

SELECT UNIX\_TIMESTAMP(NOW());

将1602389594(秒值)转换为日期

SELECT FROM\_UNIXTIME(1602389594,'%Y-%m-%d %h:%i:%s');

# 查询出生日期大于997804800的学生姓名以及出生日期

# 要求：出生日期以时间戳显示

SELECT NAME,UNIX\_TIMESTAMP(birthdate)

FROM students

WHERE UNIX\_TIMESTAMP(birthdate)<997804800;

# 进阶：显示当前时间，以及30分钟后的时间。

SELECT SYSDATE() AS 当前时间,

FROM\_UNIXTIME(UNIX\_TIMESTAMP(SYSDATE())+1800,'%Y-%m-%d %H:%i:%s') AS 30分钟后的时间;

# 进阶：假如有一个订单表，如何查昨天下单的数量呢？

SELECT COUNT(\*)

FROM 订单表

WHERE 订单时间 BETWEEN STR\_TO\_DATE('2024-10-24 00:00:00','%Y-%m-%d %H:%i:%s')

AND STR\_TO\_DATE('2024-10-24 23:59:59','%Y-%m-%d %H:%i:%s');

SELECT UNIX\_TIMESTAMP('2024-10-24 00:00:00'); # 1729699200

SELECT UNIX\_TIMESTAMP('2024-10-24 23:59:59'); # 1729785599

SELECT COUNT(\*)

FROM 订单表

WHERE 订单时间 BETWEEN UNIX\_TIMESTAMP('2024-10-24 00:00:00') AND UNIX\_TIMESTAMP('2024-10-24 23:59:59');

了解**毫秒值**的处理方式：

返回当前时间（带毫秒值）

SELECT CURRENT\_TIMESTAMP(3);

将现在日期转换为时间戳(豪秒值)

SELECT REPLACE(UNIX\_TIMESTAMP(CURRENT\_TIMESTAMP(3)),'.' ,'');

将1602389594168(毫秒值)转换为日期

SELECT

FROM\_UNIXTIME(1602389594168/1000,'%Y-%m-%d %h:%i:%s');

注意: **1秒 = 1000毫秒**,所以时间戳要除以1000。

**2.4 其他函数**

MySQL中除了上述学生的几类内置函数外，还包含了很多函数，如条件判断函数，用于在SQL语句中进程条件判断。

if(条件,值1，值2 ) 用于实现简单的条件判断。

条件为真时返回值1，条件为假时返回值2。

# 判断学生是否有手机号？

SELECT IF(mobile,'有','没有') FROM students;

ifnull(v1,v2) 也是一种条件判断函数，其表示的是如果表达式v1不为空时，则返回v1的值；否则返回v2的值。

# 查询学生的手机号。

# 要求：有手机号的显示手机号，没有手机号的显示“没有”。

SELECT IFNULL(mobile,'没有') FROM students;

case when 函数是一种更复杂的条件表达式，类似于if -else语句的组合。

case

when 条件1 then 结果1

when 条件2 then 结果2

......

else 结果

end

说明：

当对应的条件成立时，对应的结果会被返回；

如果所有的比较值都不成立，则返回else的结果；

如果没有else并且所有的条件都不成立时，则返回null。

# 查看students表的学生姓名，性别。

SELECT NAME,sex

FROM students;

# 查看students表的学生姓名，性别。

# 要求：性别为男显示某某帅哥，性别为女显示某某美女。

SELECT NAME,sex,

CASE

WHEN sex='男' THEN CONCAT(LEFT(NAME,1),'帅哥')

WHEN sex='女' THEN CONCAT(LEFT(NAME,1),'美女')

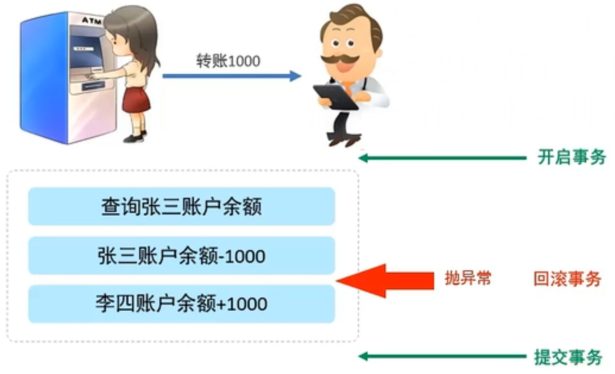
ELSE '保密'

END AS 称呼

FROM students;

### 十四、事务

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。**



为什么要有事务？

事务广泛的运用于订单系统、银行系统等多种场景。

如：张三用户和李四用户是银行的储户，现在张三要给李四转账500元，那么需要做以下几件事：

第一步：检查张三的账户余额>1000元；

第二步：张三账户中扣除>1000元；

第三步：李四账户中增加>1000元；

正常的流程走下来，张三账户扣了1000元，李四账户加了1000元，皆大欢喜。

那如果张三账户扣了钱之后，系统出故障了呢？张三白白损失了1000元，而李四也没有收到本该属于他的1000元。

以上的案例中，隐藏着一个前提条件：张三扣钱和李四加钱，要么同时成功，要么同时失败，事务的需求就在于此。

**事务命令：**

要求表的引擎类型必须是innodb类型才可以使用事务，这是mysql表的默认引擎。

修改数据时会触发事务，如：insert、update、delete。

**开启事务命令：**begin;

开启事务后执行修改命令，数据变更会维护到本地缓存中，而不维护到物理表中。

**提交事务命令：**commit;

将缓存中的数据变更维护到物理表中。

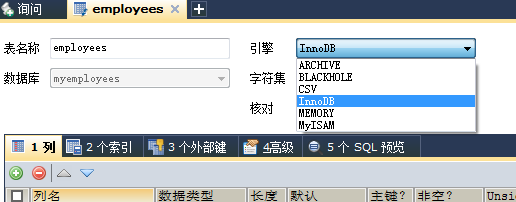
**回滚事务命令：**rollback;

放弃缓存中变更的数据。

**异常参考：**

表的引擎类型必须是 **innodb** 类型才可以使用事务。

show create table 表名;也可以看到**engine=innodb**。



**事务的语法格式：**

begin

SQL1

SQL2

......

rollback或者commit ;

**注意：**

MySQL 事务默认状态为提交状态，

oracle事务默认状态为关闭状态，必须手动提交。

SELECT @@autocommit; # 查看当前数据库事务状态。

SET @@autocommit=0; # 将事务状态设置为手动提交。

SET @@autocommit=1; # 将事务状态设置为自动提交。

# 把刘小华的年龄给王小明1岁

SET @@autocommit=0;

BEGIN

UPDATE students SET age=age-1 WHERE NAME='刘小华';

UPDATE students SET age=age+1 WHERE NAME='王小明';

# rollback;

COMMIT;

END;

**事务的特点：（又称ACID特性）**

**原子性**

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

**一致性**

事务必须使数据库从一个一致性状态变换到另外一个一致性状态。

**隔离性**

事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

**持久性**

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响。



**了解并发事务会带来哪些问题？**

在应用程序中，多个事务并发运行，经常会操作相同的数据来完成各自的任务（多个用户对同一数据进行操作）。并发虽然是必须的，但可能会导致以下的问题:

**脏读**： 当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使 用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。

**不可重复读**：指在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。

**幻读**：幻读与不可重复读类似。它发生在一个事务（T1）读取了几行数据，接着另一个并发事务（T2）插入了一些数据时。在随后的查询中，第一个事务（T1）就会发现多了一些原本不存在的记录，就好像发生了幻觉一样，所以称为幻读。

### 十五、索引

索引(index)是帮助MySQL**高效获取数据**的**数据结构**(有序)。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构,这些数据结构以某种方式引用(指向)数据，这样就可以在这些数据结构上实现高级查找算法，这种数据结构就是索引。



**思考：在字典中是如何找到一个汉字？**

一般的应用系统对比数据库的读写比例在10:1左右，而且插入操作和更新操作很少出现性能问题，遇到最多的，也是最容易出问题的，还是一些复杂的查询操作，所以查询语句的优化显然是重中之重。

当数据库中数据量很庞大时，查找数据会变得很慢。

优化方案：添加索引。

如：select \* from users where age=45;



**索引类型：**

在MySQL中索引是在存储引擎层实现的，而不是在服务器层实现的，所以不同存储引擎具有不同的索引类型和实现。

常见的索引分类：

按数据结构分类：**B+tree索引，Hash索引。**

按物理存储分类：**聚集索引，非聚集索引。**

按字段特性分类：**主键索引，唯一索引，普通索引。**

按字段个数分类：**单列索引、联合索引。**

我们学习过程中主要了解 **BTREE**（二叉树）即可。

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

**索引的优缺点：**

**优点：**

数据库查询效率优化的一种手段；相当于给每一条数据加上了标签，加快查询速度、提高查询效率。

**缺点：**

虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE，因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件。

在互联网应用中,通常查询的语句远远大于增删改的语句,甚至可以占到80%~90%,所以也不要太在意,只是在大数据导入时,可以先删除索引,再批量插入数据,最后再添加索引。

**查看索引语法格式：**

SHOW INDEX FROM 表名；

# 查看学生表索引

SHOW INDEX

FROM students;

**创建索引语法格式：**

**CREATE INDEX 索引名**

**ON 表名（字段名[长度]）;**

**注意：**

对于已经存在的表添加索引时：

如果指定字段**是字符串**，需要**指定长度**，

建议长度与定义字段时的长度一致；

如果指定字段类型**不是字符串**，可以**不填写长度**。

# 给学生表age字段添加索引。

CREATE INDEX i\_age

ON students(age);

# 给学生表name字段添加索引。

CREATE INDEX i\_name

ON students(NAME(30));

**删除索引语法格式：**

DROP INDEX 索引名

ON 表名;

# 删除学生表age字段上的索引。

DROP INDEX i\_age

ON students;

# 删除学生表name字段上的索引。

DROP INDEX i\_name

ON students;

**常见索引失效的情况有哪些？**  
1.在WHERE子句中，如果在OR前的条件列是索引列，而在OR后的条件列不是索引列，那么索引会失效。  
2.当我们使用左或者左右模糊匹配的时候，也就是like‘%xx’或者like‘%xx’这两种方式都会造成索引失效；  
3.当我们在查询条件中对索引列使用函数会导致索引失效。  
4.当我们在查询条件中对索引列进行表达式计算，也是无法走索引的。

**索引，不要总是用B树索引！**  
在数据库性能优化中，索引是一个非常关键的工具，它能够显著提高数据检索的速度。许多开发者在创建索引时，习惯性地使用B-tree索引。然而，你是否想过，单纯依赖B-tree索引可能会导致一些效率问题？  
**首先，**虽然B-tree索引可以有效地支持范围查询和排序操作，但它并不是解决所有索引问题的万能选择。在某些情况下，过度依赖B-tree索引可能会导致索引维护成本上升。例如，当数据量非常庞大，并且频繁进行插入和删除操作时，B-tree索引的更新会变得昂贵，这可能会影响整体性能。  
**其次，**使用B-tree索引的逻辑并不总是清晰。很多时候，我们需要的不仅仅是简单的查找，而是需要对特定类型的查询进行优化。这时，单纯的B-tree索引就显得有些力不从心了。例如，如果你经常进行全文搜索，那么使用全文索引会更加有效。  
**那么，如何才能在索引使用中更好地表达我们的意图呢？**  
建议根据查询类型选择合适的索引。

例如，如果你想要加速重排查询，可以使用B-tree索引：  
CREATE INDEX idx\_age ON students(age);   
但如果你需要支持全文搜索，可以使用 FULLTEXT 索引：  
CREATE FULLTEXT INDEX idx\_name ON students(NAME(30));   
这样的索引选择不仅能够提高查询效率，还能降低系统负担。  
简单来说，简单查询用B-tree索引，复杂查询用FULLTEXT索引，因为它更加高效。  
  
  
​

**了解：如何获知筛选数据时有没有使用索引进行优化呢？**

**我们可以通过 explain 命令对查询语句进行分析。**

准备数据：

在empp库中创建test\_index测试表

CREATE DATABASE empp;

CREATE TABLE test\_index(title VARCHAR(10));

创建存储过程p\_test（在存储过程中实现插入数据）。

DELIMITER //

CREATE PROCEDURE p\_test()

BEGIN

DECLARE i INT DEFAULT 1;

WHILE i<=10000 DO

INSERT INTO test\_index(title) VALUES(CONCAT('test',i));

SET i=i+1;

END WHILE;

END //

执行存储过程p\_test

实现往test\_index表的title列中添加10000条数据。

CALL p\_test();

开启监测时间

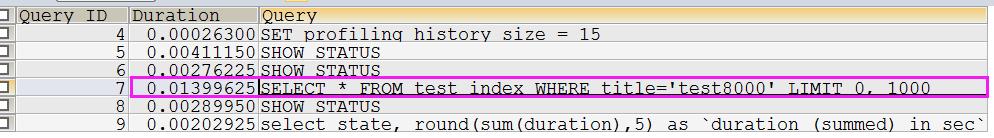
SET profiling=1;

查找第8000条数据 test8000

SELECT \* FROM test\_index WHERE title='test8000';

查看执行时间

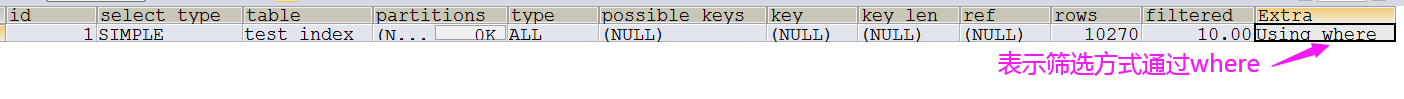
SHOW profiles;



使用 **explain** 命令分析对test\_index表的筛选方式。

EXPLAIN

SELECT \* FROM test\_index WHERE title='test8000';



给test\_index表中的title字段创建索引。

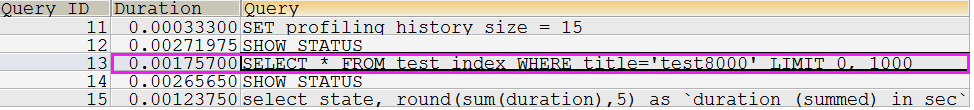
CREATE INDEX title\_index ON test\_index(title(10));

再次查找第8000条数据 test8000

SELECT \* FROM test\_index WHERE title='test8000';

再次查看执行时间

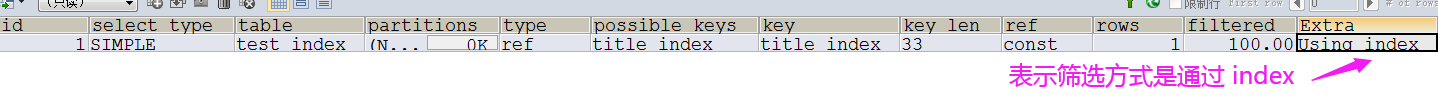
SHOW profiles;



再次使用 explain 命令分析对test\_index表的筛选方式。

EXPLAIN

SELECT \* FROM test\_index WHERE title='test8000';



**小结：**

**1、你们公司用的什么数据库？什么版本？**

**2、什么时候用到数据库？**

**3、落库数据你是怎么校验的？**

**4、常用的数据库命令有哪些？**

**5、数据的增删改查基本语句有哪些？**

**6、内连接与外连接有什么区别？**

**7、左外连接与右外连接有什么区别？**

**8、数据库的约束的用是什么？有哪些约束？**

**9、视图的作用是什么？**

**10、索引的作用是什么？有哪些索引？**

**11、事务的特点是什么？**