什么是数据库：按照数据结构来组织、存储和管理数据的仓库

关系数据库：Oracle DB2 MySQL SQL server

下载mysql

下载可视化客户端navicat

Win+R 输入service.msc可以查看/开启/关闭mysql服务

Win+R 输入cmd可以使用命令使用mysql，前提是已经配置环境变量

命令行

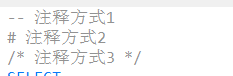
数据定义语言（DDL）：Data Definition Language，用来定义数据库的对象，如数据表、视图、索引等。

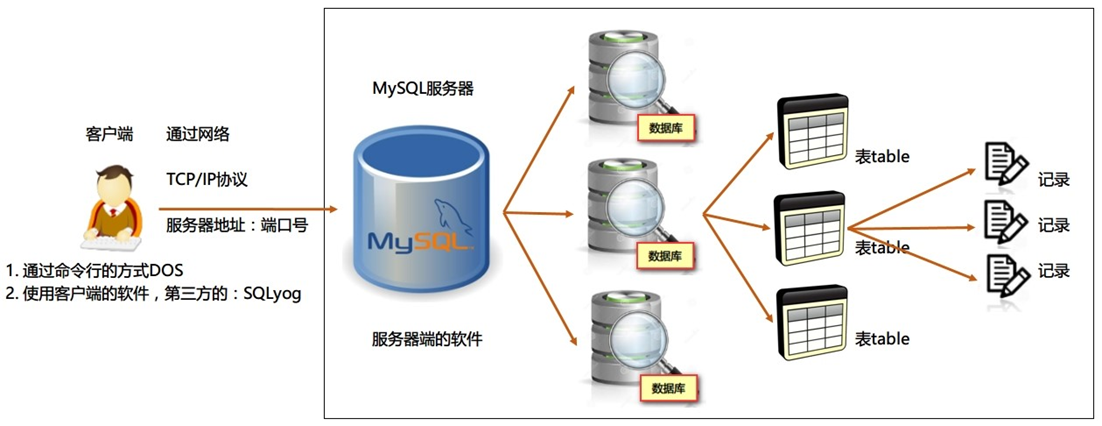
数据操纵语言（DML）：Data Manipulation Language，用来在数据库表中更新，增加和删除记录。

数据查询语言（DQL）：Data Query Language，用来查询数据库中的数据。

数据控制语言（DCL）：Data Control Language，用来控制数据库的用户权限。

事务控制语言（TCL）：Transaction Control Language，用来做数据库中的事务管理。





一个数据库服务器包含多个库。

一个数据库包含多张表。

一张表包含多条记录。

查看所有数据库

SHOW databases;

查看指定数据库的属性

SHOW CREATE DATABASE xxx;

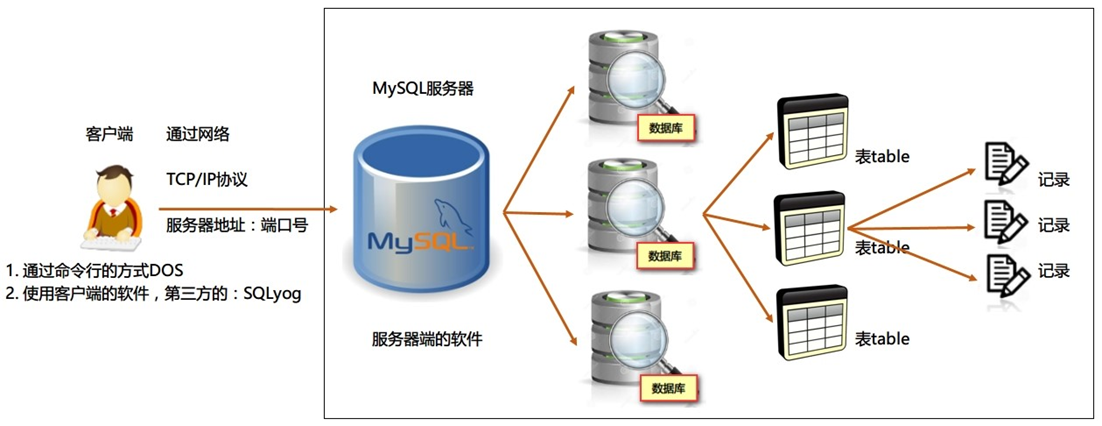
## DDL数据定义语言

### 概述

DDL(Data Definition Language):数据定义语言，用来定义数据库对象，库、表、列等；创建、删除、修改 库，表结构。主要分为操作数据库的DDL和操作表的DDL。

### 操作数据库的DDL

这里所提到的数据库不同于我们之前提到的MySQL数据库、Oracle数据库等。MySQL数据库中的”数据库”是指一个数据库管理系统（DBMS），而这里所提到的数据库是指用来组织和管理一些对象（表、视图等）的结构，简单可以理解问磁盘上的文件夹。



一个数据库服务器包含多个库。

一个数据库包含多张表。

一张表包含多条记录。

#### 创建数据库

##### 直接创建数据库

**语法**

**CREATE DATABASE [IF NOT EXISTS]** 数据库名;

**示例**

CREATE DATABASE mydb1;

或

CREATE DATABASE IF NOT EXISTS mydb2;

##### 指定字符集方式创建数据库

**语法**

**CREATE DATABASE [IF NOT EXISTS]** 数据库名 **CHARACTER SET** 字符集

**示例**

CREATE DATABASE mydb3 CHARACTER SET UTF8;

##### 指定字符集和排序规则方式创建数据库

**语法**

**CREATE DATABASE[IF NOT EXISTS]** 数据库名 **CHARACTER SET** 字符集 **COLLATE** 排序规则

**示例**

CREATE DATABASE mydb4 CHARACTER SET UTF8 COLLATE utf8\_bin

**排序规则说明**

针对不同字符集，MySQL指定了不同的排序规则（可以参考MySQL的帮助文档）。

例如：utf8字符集中指定的 **utf8\_general\_ci** 和 **utf8\_bin**，对数据库中存储数据库的规则就有所不同。

**utf8\_general\_ci**：ci是 case insensitive, 即 "大小写不敏感", a 和 A 会在字符判断中会被当做一样的。

**utf8\_bin**：将字符串每个字符串用二进制数据编译存储，区分大小写，而且可以存二进制的内容。

#### 查看数据库

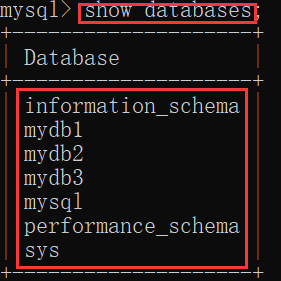
##### 查看所有数据库

**语法**

SHOW databases;

**示例**

SHOW databases;



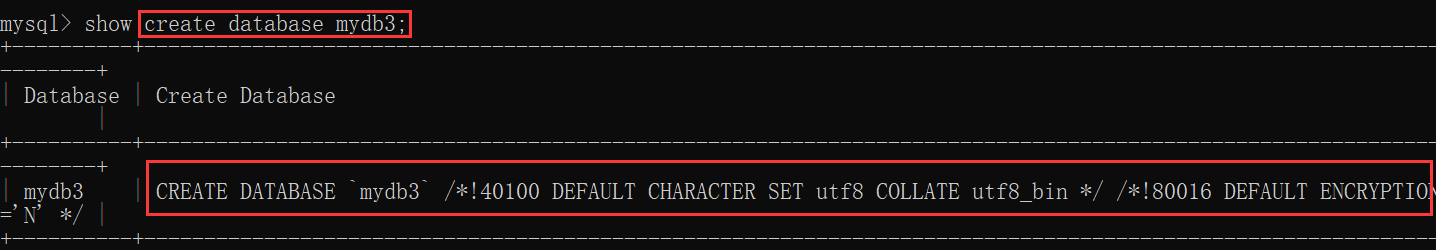
##### 查看指定数据库的创建属性

**语法**

SHOW CREATE DATABASE 数据库名;

**示例**

SHOW CREATE DATABASE mydb3



#### 修改数据库

只能修改数据库的字符集和排序规则，不能修改数据库名字

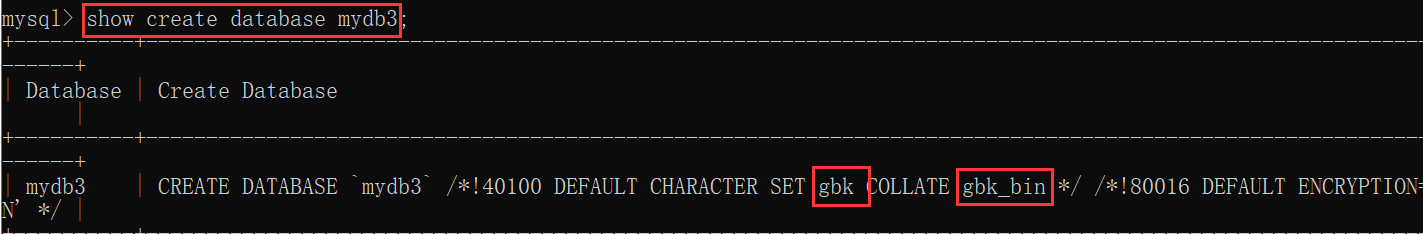
**语法**

**ALTER DATABASE** 数据库名 **CHARACTER SET** 字符集 [**COLLATE** 排序规则]

**示例**

修改mydb3的字符集为gbk 排序规则为 gbk\_bin

alter database mydb3 character set gbk collate gbk\_bin;



#### 删除数据库

**语法**

DROP DATABASE 数据库名

**示例**

删除mydb3数据库

DROP DATABASE mydb3

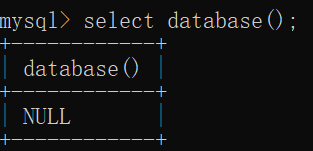
#### 使用数据库

##### 查看正在使用的数据库

**语法**

SELECT DATABASE();

**示例**



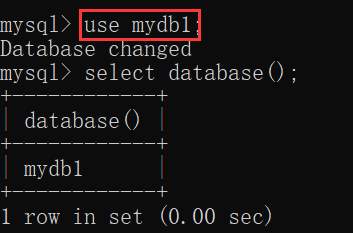
##### 切换数据库

语法

USE 数据库名;

示例

USE mydb1



### 操作表的DDL

表是一种数据库对象,表由若干个字段(列)组成。是我们在操作数据库时使用最为频繁的数据库对象。

#### 创建表

**注意：在创建表之前，一定要先使用数据库（也就是说，表一定要创建在某个数据库中）**

**语法**

CREATE TABLE 表名 (字段名1 数据类型,字段名2 数据类型,…,字段名n 数据类型)

建议写成如下格式:

CREATE TABLE 表名 (  
字段名1 数据类型,   
字段名2 数据类型,

...,

字段名n 数据类型

)

**示例**

创建student表包含id,name,age,sex字段

CREATE TABLE student (  
 id INT,  
 name VARCHAR(20),  
 birthday date  
 )

#### 数据类型

##### 数值类型

###### 整数类型

|  |  |  |
| --- | --- | --- |
| **类型名称** | **字节** | **说明** |
| tinyint | 1 | 无符号：0~255 有符号：-128~127 |
| smallint | 2 | 无符号：0~65535 有符号：-32768~32767 |
| mediumint | 3 | 无符号：0~1677215 有符号：-8388608~8388607 |
| **int、**integer | 4 | 无符号：0~4294967295 有符号：-2147483648~2147483647 |
| bigint | 8 | 无符号：0~18446744073709551615  有符号：-9223372036854775808~9223372036854775807 |

**在建表时默认都是有符号的。**

**如要使用无符号的，在建表语句中指定。例如：id int unsigned**

###### 近似数类型

在MySQL中，可以通过浮点数和定点数来表示近似值。

**浮点数**

MySQL中的浮点数包括float和**double**两种，定义格式为float(M,D)、double(M,D)。

M-表示所存储的值共有M位(M的取值范围是0~255)

D-表示小数点后有D位(D的取值范围是0~30)

例如:float(7,4)存储的数据范围时-999.9999~999.9999。

**定点数**

decimal是MySQL的定点数。相较与浮点数定点数表示的近似值更加精确。定义格式为decimal(M,D)

M-表示所存储的值共有M位(M的默认取值是10，取值范围是1~65)

D-表示小数点后有D位(D的默认取值是0，取值范围是0~30)

##### （2）字符串类型

MySQL提供了多种针对字符串的存储类型。如下表所示

|  |  |
| --- | --- |
| **字符串类型** | **取值范围和存储需求** |
| char(M) | M为0~255之间的整数 |
| **varchar(M)** | M为0~65535之间的整数 |
| tinyblob | 允许长度0~255字节 |
| blob | 允许长度0~65535字节 |
| meduimblob | 允许长度0~167772150字节 |
| longblob | 允许长度0~4294967295字节 |
| tinytext | 允许长度0~255字节 |
| text | 允许长度0~65535字节 |
| mediumtext | 允许长度0~167772150字节 |
| longtext | 允许长度0~4294967295字节 |
| varbinary(M) | 允许长度0~M个字节的变长字节字符串 |
| binary(M) | 允许长度0~M个字节的定长字节字符串 |
| enum | 允许的值选自一个允许值列表，这个列表最多能存储的值数量为65535个 |
| set | 和enum类似，set类型是一个多选字符串数据类型 |

###### a、char和varchar

char和varchar两种类型相似，都是MySQL数据库中用来存储较短字符串格式的类型。但这两种类型的主要区别在于存储方式不同，char类型的长度是固定的，而varchar类型的长度是可变的。例如，存储字符串”abc”，char(5)表示存储的字符将占5个字节（包括2个空字符），而varchar(5)只占用3个字节的长度，5只是最大值，当存储的字符小于5个时，按实际长度存储。同时在检索时，char类型会删除尾部的空格，而varchar类型会保留尾部的空格。**因为长度的问题，所以导致两种数据类型的数据寻址速度不同，如果字符在10个以内，推荐使用char。**

###### b、BLOB和TEXT

BLOB是一个二进制大对象，可以容纳可变数量的数据。有4种BLOB类型：**tinyblob**、**blob**、**mediumblob**和**longblob**。它们只是可容纳值的最大长度不同。

有4种TEXT类型：**tinytext、text、mediumtext**和**longtext**。这些对应4种BLOB类型，有相同的最大长度和存储需求。

因为大文件中的字符可能存在非法，所以推荐使用BLOB，比如ascii字符集下存储一张含有”0XFF”字节的图片，入库过滤。

##### （3）日期和时间类型

MySQL中有多种日期和时间类型的数据数据类型。如下表所示

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **字节** | **最小值** | **最大值** |
| **Date** | 4 | 1000-01-01 | 9999-12-31 |
| Datetime | 8 | 1000-01-01 00:00:00 | 9999-12-31 23:59:59 |
| Timestamp | 4 | 1970-01-01 00:00:01.000000 | 2038-01-19 03:14:07.999999' |
| Time | 3 | -838:59:59 | 838:59:59 |
| Year | 1 | 1901 | 2155 |

对于这些数据类型的选择，有如下使用准则：

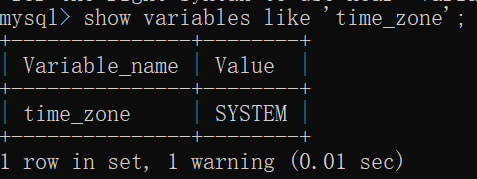
1. 如果要表示年月日，通常使用date。
2. 如果要表示年月日时分秒，通常使用datetime。
3. 如果要表示时分秒，通常使用time。
4. 如果要表示年，通常使用year。
5. 如果要表示当前系统的时间，通常使用timestamp。

**timestamp和datetime**

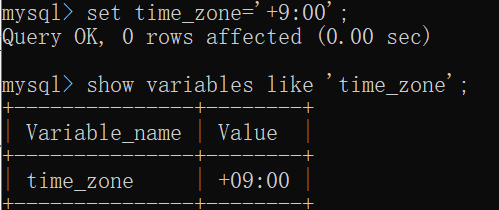
虽然timestamp类型和datetime类型一样可以表示日期时间，但两者存在差别。

1、timestamp类型存储的时间与MySQL数据库系统安装时所选的时区有关，在不同时区下查看的同一批数据将会得出不同的结果。

**查看时区的SQL语句：** **show variables like ‘time\_zone’; 默认与主机一致，显示结果为SYSTEM。**



**修改时区的SQL语句：set time\_zone=’+9:00’(设置为东9区)**



1. 两者的取值范围不同，datetime的取值范围是1000-01-01 00:00:00到9999-12-31 23:59:59，timestamp的取值范围是1970-01-01 00:00:01.000000到2038-01-19 03:14:07.999999'。

最常用类型总结：整数 int 、 浮点数 double 、字符串 varchar、日期 date。

**创建表示例**

需求：创建employee表，包含字段，编号(id int)、姓名(emp\_name varchar)、性别(gender enum(‘M’,’F’))、入职日期(hire\_date date)、薪资(salary double)、简历(resume text)

create table employee(

id int,

emp\_name varchar(100),

gender enum(‘M’,’F’),

hire\_date date,

salary double,

resume text

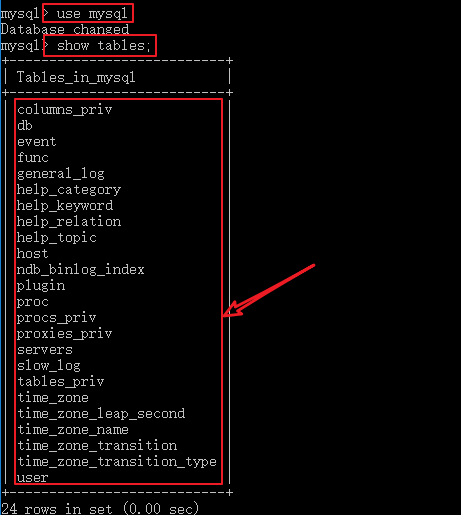
)

##### 3、查看表

##### （1）查看所有表

**语法**

show tables



##### 查看表结构

**语法**

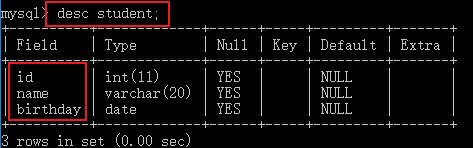
DESC 表名;

或

describe 表名

**示例**

desc student



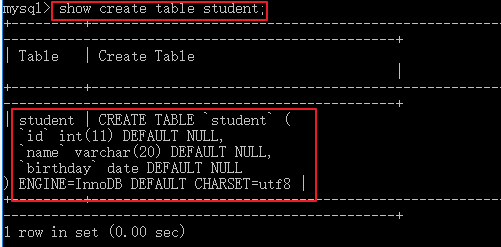
##### 查看表的创建属性

**语法**

SHOW CREATE TABLE 表名;

**示例**

show create table student



#### 4、复制表结构

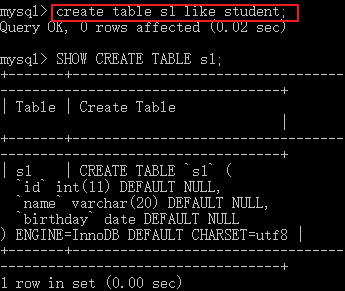
**语法**

CREATE TABLE 新表名 LIKE 旧表名

**示例**

创建s1表，s1表结构和student表结构相同

CREATE TABLE s1 LIKE student;



#### 修改表

##### 添加新列(字段)

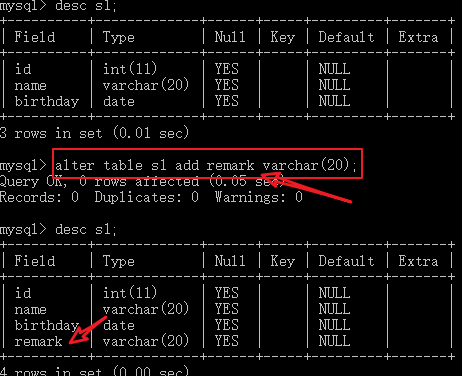
**语法**

ALTER TABLE 表名 ADD 列名 类型

**示例**

为学生表s1添加一个新的字段remark，类型为varchar(20)

ALTER TABLE s1ADD remark VARCHAR(20);



##### 修改列字段类型

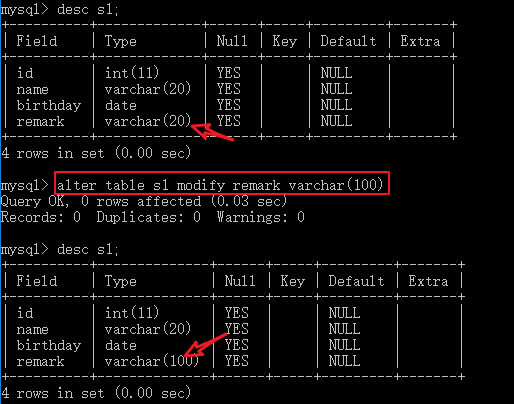
**语法**

ALTER TABLE 表名 MODIFY 列名 新的类型

**示例**

将s1表中的remark字段的改成varchar(100)

ALTER TABLE s1MODIFY remark VARCHAR(100)



##### 修改列名

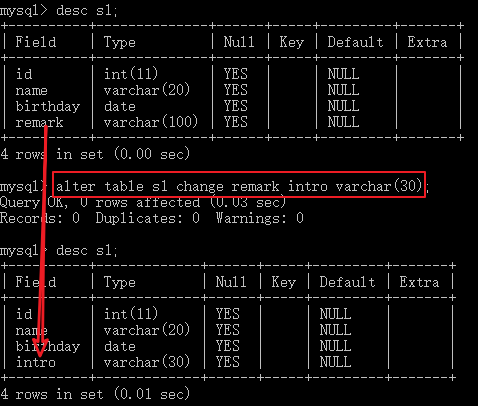
**语法**

ALTER TABLE 表名 CHANGE 旧列名 新列名 类型

**示例**

将s1表中的remark字段名改成intro，类型varchar(30)

ALTER TABLE s1 CHANGE remark intro varchar(30)



##### 删除列

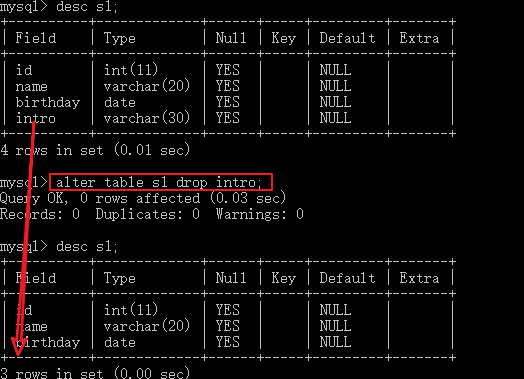
**语法**

ALTER TABLE 表名 DROP 列名

**示例**

删除s1表中的字段intro

ALTER TABLE s1 DROP intro



##### 修改表名

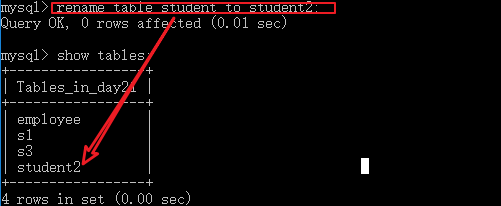
**语法**

RENAME TABLE 表名 TO 新表名

**示例**

将学生表student改名成student2

RENAME TABLE student TO student2



##### 修改字符集

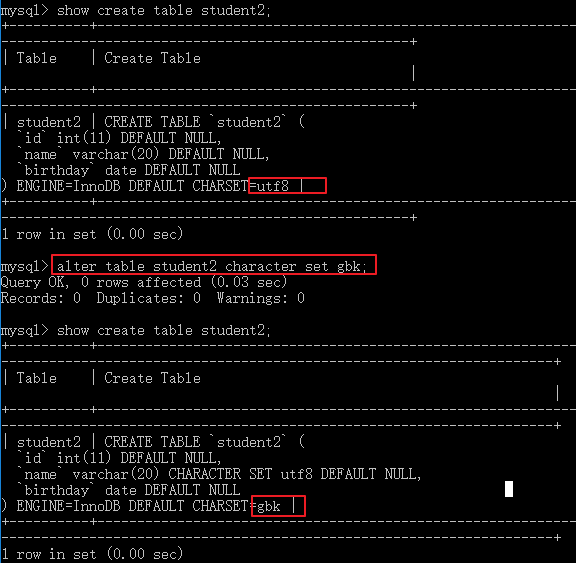
**语法**

ALTER TABLE 表名 character set 字符集;

**示例**

将sutden2表的编码修改成gbk

ALTER TABLE student2 character set gbk



#### 删除表

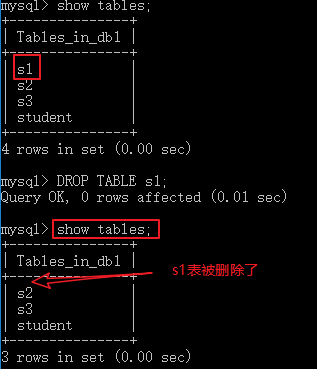
**语法**

DROP TABLE 表名;

**示例**

删除s1表

drop table s1



## DML数据操纵语言

对表中的数据进行增、删、改操作。

### 插入记录

#### 指定字段的插入

**语法**

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

**示例**

更换数据库，创建student表

create table student(

id int,

name varchar(20),

age int,

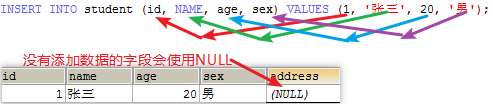
sex char(1),

address varchar(100)

)

向学生表中添加 id, name, age, sex数据

INSERT INTO student (id, name, age, sex) VALUES (1, '张三', 20, '男');



**示例2**

向student表中插入数据

INSERT INTO student (NAME, id, age, sex, address) VALUES ('李四', 2, 23, '女', '广州');



**注意事项**

值与字段必须对应，个数相同，类型相同，顺序可以不严格按照创建表时的字段顺序

值的数据大小必须在字段的长度范围内 varchar()

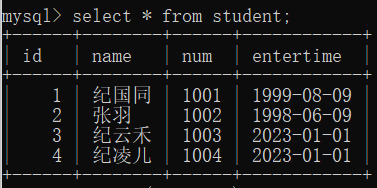
除了数值类型外，其它的字段类型的值必须使用引号引起（建议单引号）

如果要插入空值，可以不写字段，或者插入null

**insert into student (id,name,num,entertime) values(01,'纪国同',1001,'1999-08-09');**

**insert into student (id,name,num,entertime) values;**

**insert into student (id,name,num,entertime) values (02,'张羽',1002,'1998-06-09') ,(03,'纪云禾',1003,'2023-01-01'), (04,'纪凌儿',1004,'2023-01-01');**



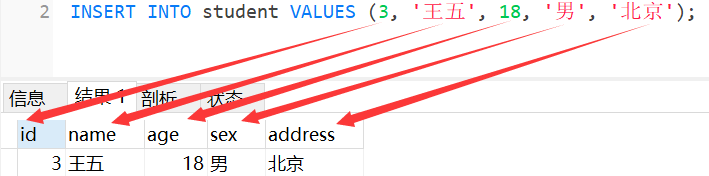
#### 2、不指定字段的插入

**语法**

INSERT INTO 表名 VALUES (值1, 值2…)

**示例**

INSERT INTO student VALUES (3, '王五', 18, '男', '北京');



**注意事项**

不指定列时，相当于指定了所有列，顺序和建表时的顺序一致。

#### 3、蠕虫复制

在已有的数据基础之上，将原来的数据进行复制，插入到对应的表中。

**语法**

INSERT INTO 表名1 SELECT \* FROM 表名2;

**示例**

创建student2表，student2结构和student表结构一样

CREATE TABLE student2 LIKE student;

将student表中的数据添加到student2表中

INSERT INTO student2 SELECT \* FROM student

### 更新表记录

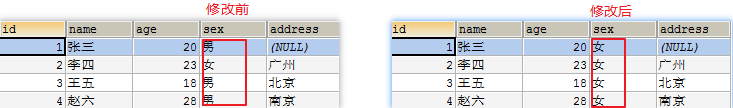
#### 不带条件的更新——针对所有行

**语法**

UPDATE 表名 SET 字段1=值1[,字段2=值2,,...,字段n=值n]

**示例**

UPDATE student SET sex='女'



**说明**

表中所有行的sex都会被更新为女

#### 带条件的更新——针对符合条件的行的值

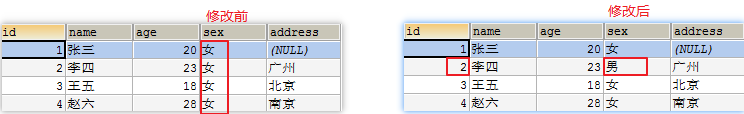
**语法**

UPDATE 表名 SET 字段1=值1[,字段2=值2,,...,字段n=值n][where 条件]

**示例**

将id号为2的学生性别改成男

UPDATE student SET sex='男' WHERE id=2;



**说明**

where 条件有很多种，在后面的查询中逐个讲解。

### 删除记录

#### 带条件删除

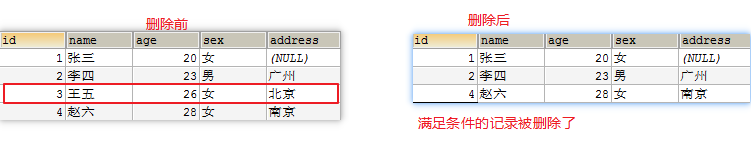
**语法**

DELETE FROM 表名 WHERE 条件

**示例**

带条件删除数据，删除id为3的记录

DELETE FROM student WHERE id=3



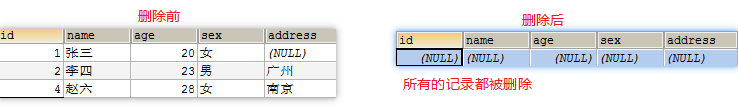
#### 不带条件删除

**语法**

DELETE FROM 表名

**示例**

DELETE FROM student;



**说明**

不带条件的删除，将删除表中的全部数据。

## DQL数据查询语言

DQL语句的作用是查询数据库中的数据。查询不会对数据库中的数据进行修改，只是一种显示数据的方式。

### 单表查询

#### 简单查询

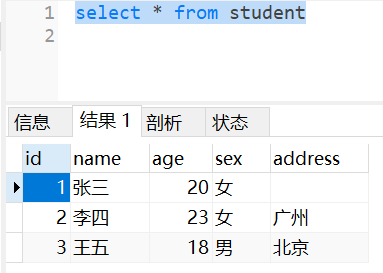
##### 查询表中全部数据

**语法一**

SELECT \* FROM 表名

**示例**

SELECT \* FROM student



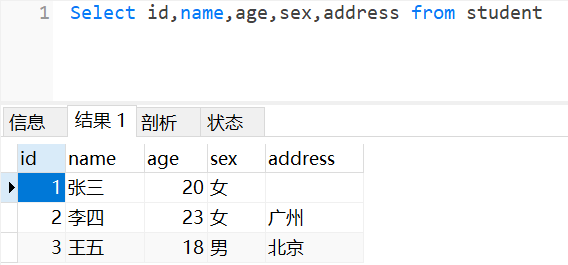
**语法二**

SELECT 字段名1, 字段名2, ...字段n FROM 表名

写出所有字段名

**示例**

Select id,name,age,sex,address from student



##### 查询指定列

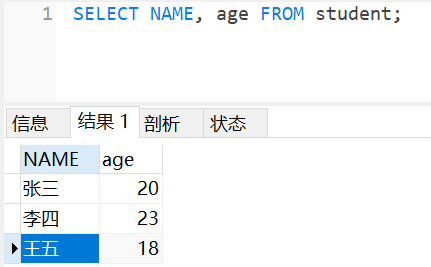
**语法**

SELECT 字段名1, 字段名2... FROM 表名

**示例**

查询student表中的name 和 age 列

SELECT name, age FROM student



没有查询的列的数据不会显示

##### 别名查询

在查询时，给查询出来的列或表起一个其他的（一般是稍短的名字，或为了防止重复）名字就叫做别名。使用别名的好处是方便查看和处理查询到的数据。

**语法**

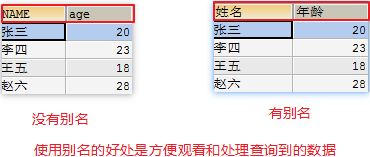
SELECT 字段名1 AS 别名, 字段名2 AS 别名... FROM 表名;

SELECT 字段名1 AS 别名, 字段名2 AS 别名... FROM 表名 AS 表别名

**示例**

查询sudent表中name 和 age 列，name列的别名为”姓名”，age列的别名为”年龄”

SELECT NAME AS '姓名', age AS '年龄' FROM student;



查询sudent表中name和age列，student表别名为s

SELECT NAME, age FROM student AS s;



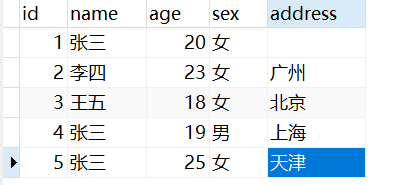
**说明**

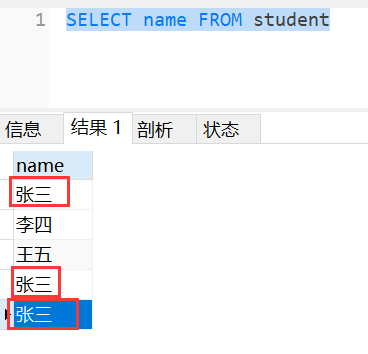
* 查询时给表取别名目前还看不到效果，需要到多表查询的时候才能体现出其好处。
* **AS关键字可以省略。**

##### 去除重复值

在查询指定列时，如果指定的列中有重复的数据，默认都会显示出来。

**例如**





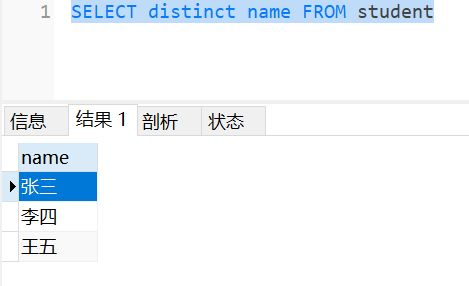
可以使用**distinct**关键字来去除重复数据

**语法**

SELECT DISTINCT 字段名 FROM 表名

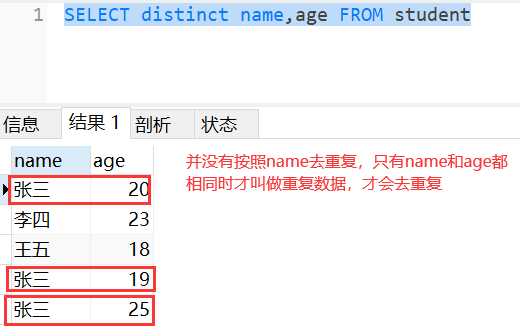
**示例**

SELECT DISTINCT name FROM student



**说明**

当distinct后出现了多个列时，只有当多个列的数据都重复才叫重复数据。



##### 查询结果参与运算

在查询语句中，查询的列的可以和其他值做数学运算(加、减、乘、除等)。运算结果只会影响展示，不会影响表中的数据。

**语法**

SELECT 列名1 + 固定值 FROM 表名

SELECT 列名1 + 列名2 FROM 表名

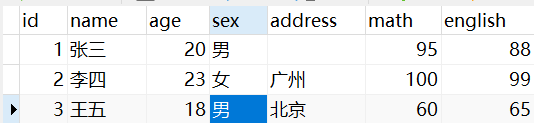
**示例**

在student表中添加数学，英语成绩列，给每条记录添加对应的数学和英语成绩

修改student表结构，添加数学和英语成绩列

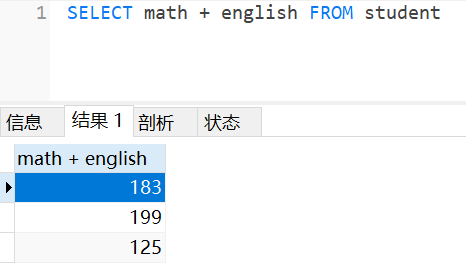
ALTER TABLE student ADD math INT;

ALTER TABLE student ADD english INT



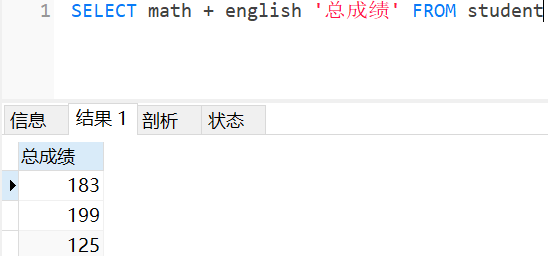
**查询math + english的和**

SELECT math + english FROM student



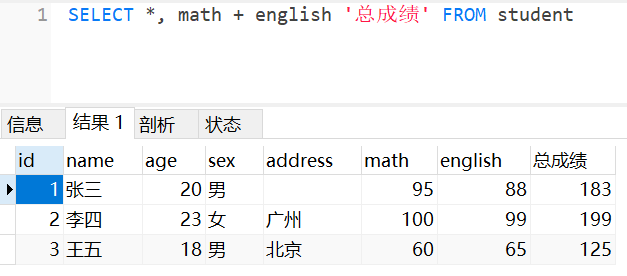
**查询math + english的和，并使用别名”总成绩”**

SELECT math + english '总成绩' FROM student



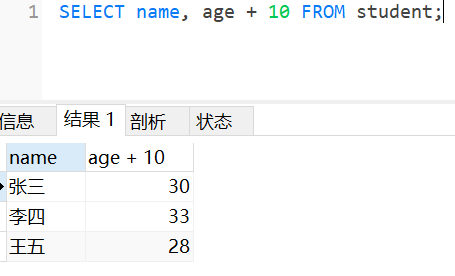
**查询所有列与math + english的和，并使用别名”总成绩”**

SELECT \*, math + english '总成绩' FROM student



**查询姓名、年龄，将每个人的年龄增加10岁**

**SELECT name, age + 10 FROM student**



#### 条件查询

##### 准备表和数据

CREATE TABLE student3 (  
 id int,  
 name varchar(20),  
 age int,  
 sex varchar(5),  
 address varchar(100),  
 math int,  
 english int  
 );  
  
INSERT INTO student3(id,NAME,age,sex,address,math,english) VALUES (1,'马云',55,'男','杭州',66,78),(2,'马化腾',45,'女','深圳',98,87),(3,'马景涛',55,'男','香港',56,77),(4,'柳岩',20,'女','湖南',76,65),(5,'柳青',20,'男','湖南',86,NULL),(6,'刘德华',57,'男','香港',99,99),(7,'马德',22,'女','香港',99,99),(8,'德玛西亚',18,'男','南京',56,65);

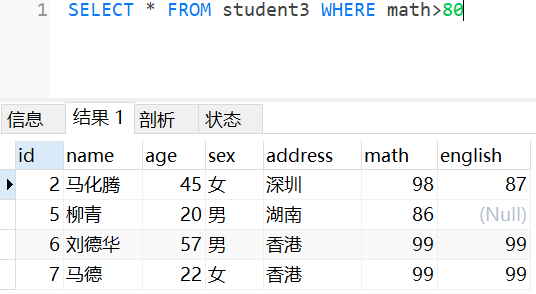
##### 比较运算符

在查询条件中，可以使用多种比较运算符来表示查询条件。

=等于、 >大于、 <小于、 <=小于等于、 >=大于等于、 <>或!=不等于

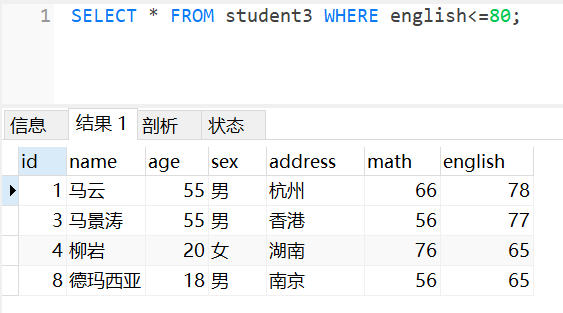
**示例1：查询math分数大于80分的学生**

SELECT \* FROM student3 WHERE math>80



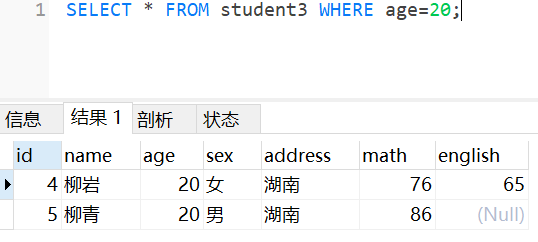
**示例2：查询english分数小于或等于80分的学生**

SELECT \* FROM student3 WHERE english<=80;



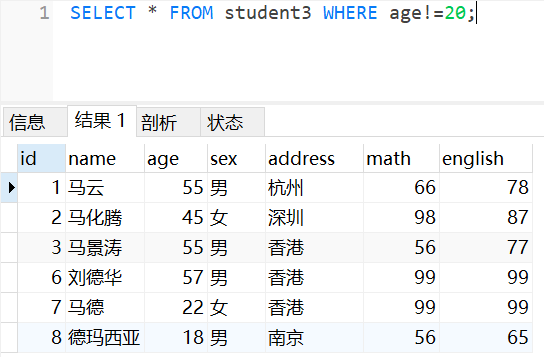
**示例3：查询age等于20岁的学生**

SELECT \* FROM student3 WHERE age=20;



**示例4：查询age不等于20岁的学生**

SELECT \* FROM student3 WHERE age!=20;  
SELECT \* FROM student3 WHERE age<>20;



##### 逻辑运算符

and(&&) 多个条件同时满足

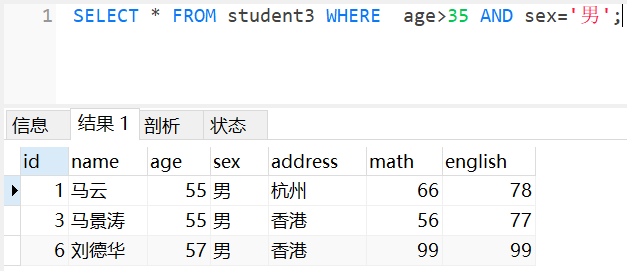
or(||) 多个条件其中一个满足

not(!) 不满足

注：有多个逻辑运算符同时使用的时候， 可以使用扩号加以规范和约束。

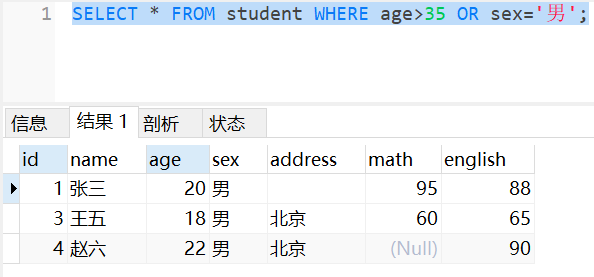
**示例1：查询age大于35且性别为男的学生(两个条件同时满足)**

SELECT \* FROM student3 WHERE age>35 AND sex='男';



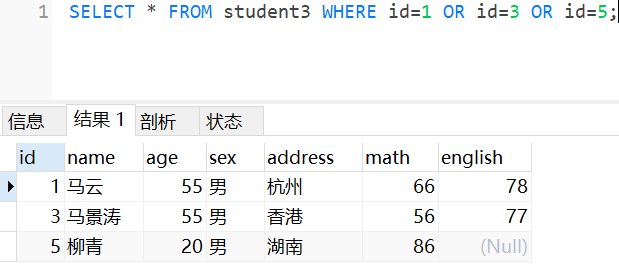
**示例2：查询age大于35或性别为男的学生(两个条件其中一个满足)**

SELECT \* FROM student WHERE age>35 OR sex='男';



**示例3：查询id是1或3或5的学生**

SELECT \* FROM student3 WHERE id=1 OR id=3 OR id=5;



**关于not，在后面的案例中讲解！**

##### 指定范围内查询 in

**语法**

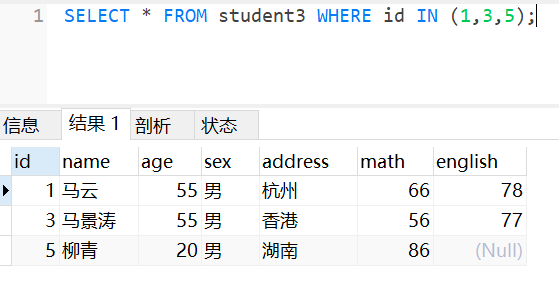
SELECT 字段名 FROM 表名 WHERE 字段 in (数据1, 数据2...);

in里面的每个数据都会作为一次条件，只要满足条件的就会显示。

**示例**

查询id是1或3或5的学生

SELECT \* FROM student3 WHERE id IN (1,3,5)



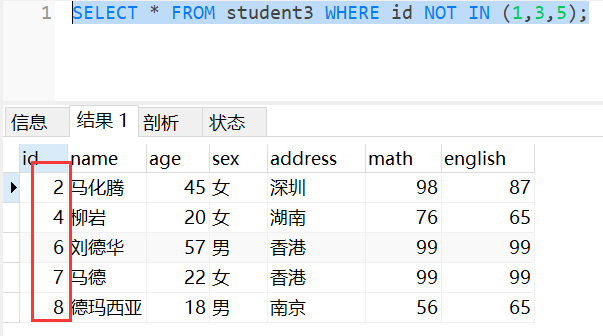
**说明**

相当于使用or连接多个条件。

**示例2**

查询id不是1或3或5的学生

SELECT \* FROM student3 WHERE id **NOT** IN (1,3,5);



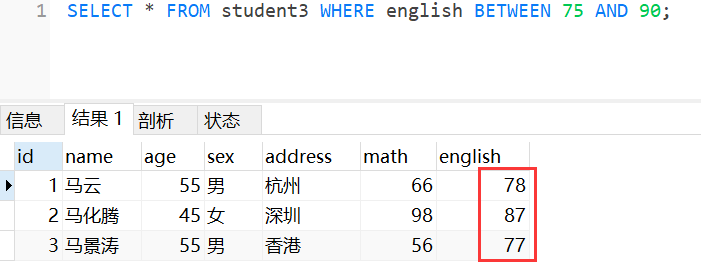
##### 范围between

BETWEEN 值1 AND 值2，表示从值1到值2范围，包括开头值又包括结尾值，闭区间

比如：age BETWEEN 80 AND 100 相当于： age>=80 and age<=100

**示例：查询english成绩大于等于75，且小于等于90的学生**

SELECT \* FROM student3 WHERE english>=75 AND english<=90;  
SELECT \* FROM student3 WHERE english BETWEEN 75 AND 90;



##### 模糊查询like

**语法**

SELECT \* FROM 表名 WHERE 字段名 like '通配符字符串';  
满足通配符字符串规则的数据就会显示出来。

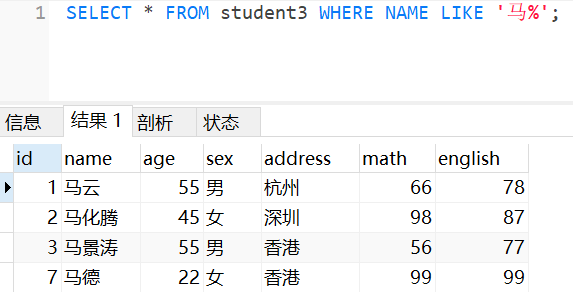
MySQL通配符有两个：

%: 表示0个或多个字符(任意个字符)

\_: 表示一个字符

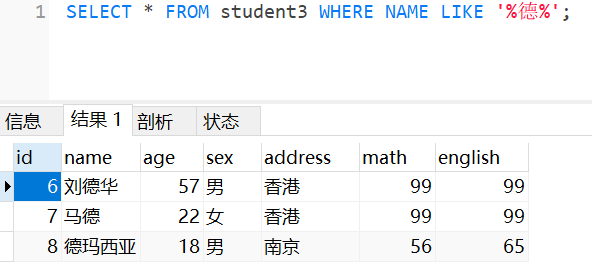
**示例1：查询姓马的学生**

SELECT \* FROM student3 WHERE NAME LIKE '马%';



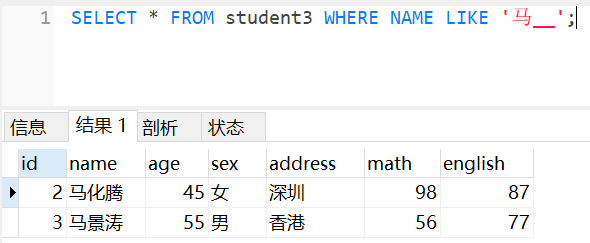
**示例2：查询姓名中包含'德'字的学生**

SELECT \* FROM student3 WHERE NAME LIKE '%德%';



**示例3：查询姓马，且姓名有三个字的学生**

SELECT \* FROM student3 WHERE NAME LIKE '马\_\_';



##### 为空查询 is null

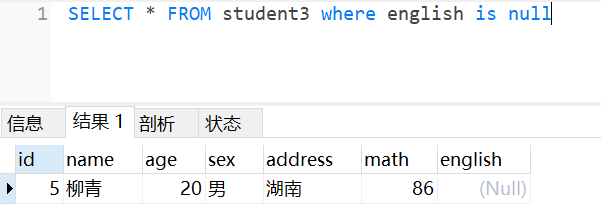
查询某个字段为空（null）的数据，不是使用=null（null和任何值都不相等），而是使用is null来进行判断；如果是筛选非空数据，使用is not null

select device\_id,gender,age,university from user\_profile where age is not null;

**示例**

查询english成绩时null的学生信息

select \* from student3 where english is null



### （二）排序

通过ORDER BY子句，可以将查询出的结果进行排序，排序只影响显示结果，不会影响数据库中数据的顺序。

**语法**

SELECT 字段名 FROM 表名 [WHERE 条件] ORDER BY 字段名 [ASC|DESC];

ASC: 升序，默认是升序

DESC: 降序

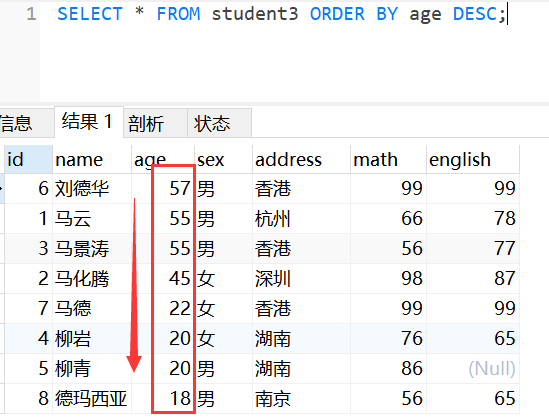
#### 单列排序

单列排序就是使用一个字段排序。

**示例**

查询所有数据，使用年龄降序排序：

SELECT \* FROM student3 ORDER BY age DESC;



#### 组合排序

组合排序就是先按第一个字段进行排序，如果第一个字段相同，才按第二个字段进行排序，依次类推。 上面的例子中，年龄是有相同的。当年龄相同再使用math进行排序。

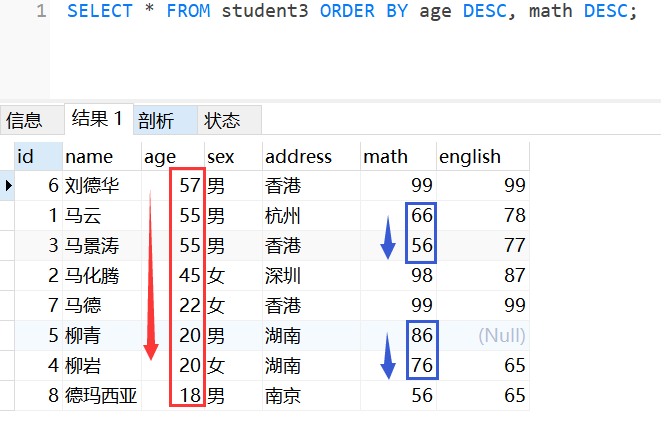
**语法**

SELECT 字段名 FROM 表名 [WHERE 条件] ORDER BY 字段名1 [ASC|DESC],字段名2 [ASC|DESC];

**示例**

查询所有数据，在年龄降序排序的基础上，如果年龄相同再以数学成绩降序排序：

SELECT \* FROM student3 ORDER BY age DESC, math DESC;



### （三）单行函数

单行函数是指对于每一行数据进行计算后得到一行输出结果。SQL单行函数根据数据类型分为字符函数、数字函数、日期函数、转换函数等。

#### 数值函数

##### abs(x)

返回 x 的绝对值

**示例**

SELECT ABS(-1) -- 返回1

##### ceil(x)

返回大于或等于 x 的最小整数（向上取整）

**示例**

SELECT CEIL(1.5) -- 返回2

##### floor(x)

返回小于或等于 x 的最大整数（向下取整）

**示例**

SELECT FLOOR(1.5) -- 返回1

##### rand()

返回 0 到 1 的随机数

**示例**

SELECT RAND() --0.93099315644334

##### round(x)

返回离 x 最近的整数(四舍五入取整)

**示例**

SELECT ROUND(1.23456) --返回1

#### 字符串函数

##### concat(s1,s2...sn)

字符串 s1,s2 等多个字符串合并为一个字符串

**示例**

SELECT CONCAT("SQL ", "Runoob ", "Google ", "Facebook") AS result;

返回 SQL Runoob Gooogle Facebook

##### locate(s1,s)

从字符串 s 中获取 s1 的开始位置

**示例**

SELECT LOCATE('st','myteststring'); -- 5

##### lower (s)

将字符串 s 的所有字母变成小写字母

**示例**

SELECT LOWER('RUNOOB') -- runoob

##### upper(s)

将字符串转换为大写

**示例**

SELECT UPPER("runoob"); -- RUNOOB

##### replace(s,s1,s2)

将字符串 s2 替代字符串 s 中的字符串 s1

**示例**

SELECT REPLACE('abc','a','x') --xbc

##### substring(s, start, length)

从字符串 s 的 start 位置截取长度为 length 的子字符串

**示例**

SELECT SUBSTRING("RUNOOB", 2, 3) AS ExtractString; -- UNO

SELECT SUBSTRING(name1,1,2) from info; #字符串位置从1开始截取前两个

##### trim(s)

去掉字符串 s 开始和结尾处的空格

**示例**

SELECT TRIM(' RUNOOB ') AS TrimmedString;

##### reverse(s)

反转字符串

**示例**

SELECT reverse(' dog'); -- 返回god

#### 时间日期函数

##### now()

返回系统当前时间(包括年月日时分秒)

**示例**

SELECT now()

##### sysdate()

返回系统当前时间(包括年月日时分秒)

**示例**

SELECT sysdate()

##### curdate()

返回当前日期(年月日)

**示例**

select curdate()

##### curtime()

返回当前时间(时分秒)

**示例**

select curtime()

##### month(date)

返回参数日期的月份值

**示例**

select MONTH('2020-11-12')

##### week(date)

返回参数日期是一年中的第几个星期

**示例**

select week('2020-11-12')

##### day(date)

返回参数日期的日值

**示例**

select day('2020-11-12')

##### date\_add(date,INTERVAL expr type)

在原有参数日期上增加相应的时间间隔

date 参数是合法的日期表达式。expr 参数是您希望添加的时间间隔。

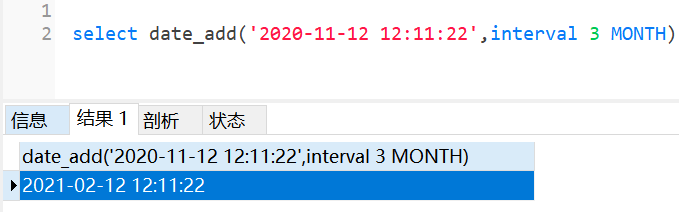
type 参数可以是下列时间类型值：

|  |
| --- |
| MICROSECOND |
| SECOND |
| MINUTE |
| HOUR |
| DAY |
| WEEK |
| MONTH |
| QUARTER |
| YEAR |
| SECOND\_MICROSECOND |
| MINUTE\_MICROSECOND |
| MINUTE\_SECOND |
| HOUR\_MICROSECOND |
| HOUR\_SECOND |
| HOUR\_MINUTE |
| DAY\_MICROSECOND |
| DAY\_SECOND |
| DAY\_MINUTE |
| DAY\_HOUR |
| YEAR\_MONTH |

SELECT date\_add(‘2023-01-01 12:00:00’,interval 6 month);

**示例**

select date\_add('2020-11-12 12:11:22',interval 3 MONTH)



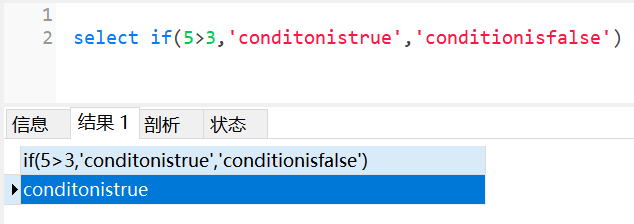
#### 流程控制函数

##### if(condition,expr1,expr2)

condition条件为true时返回expr1。condition条件为false时，返回expr2。

**示例**

select if(5>3,'conditonistrue','conditionisfalse')



将用户划分为25岁以下和25岁及以上两个年龄段，分别查看这两个年龄段用户数量

select

if(age<25 or age is null,'25岁以下','25岁及以上') as age\_cut,

count(\*) as number

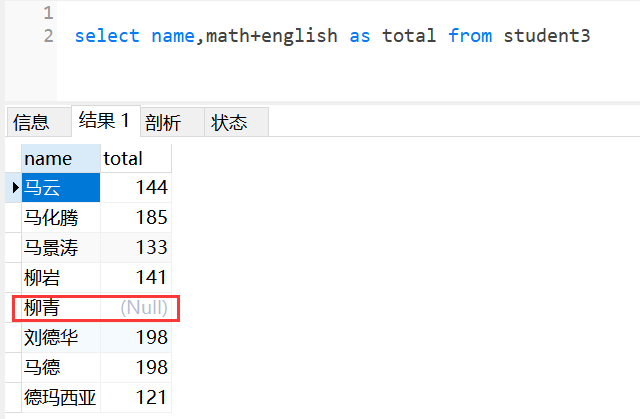
from user\_profile group by age\_cut;

##### ifnull(expr1,expr2)

expr1不为null时返回expr1。expr1为null时返回expr2。

**示例**

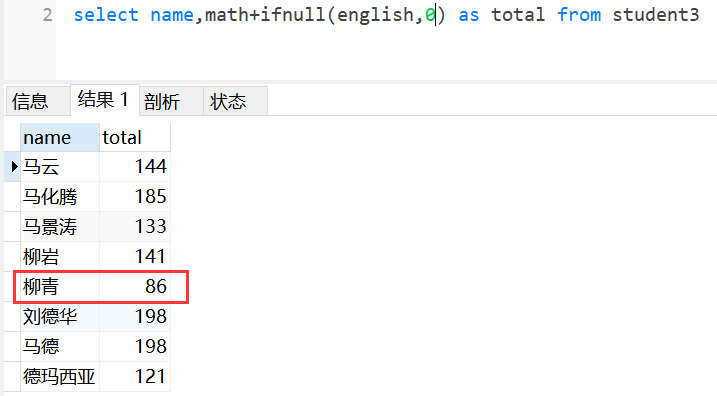
用此函数解决前面null运算结果为null的问题



解决方式

因为english有null值，所以用ifnull函数处理english

select name,math+ifnull(english,0) as total from student3



##### case when

SELECT device\_id,gender,

(CASE

    WHEN age<20 THEN '20岁以下'

    WHEN age>=20 and age<=24 THEN '20-24岁'

    WHEN age>=25 THEN '25岁及以上'

    ELSE '其他' END) age\_cut

FROM user\_profile;

#### 5、文本函数

（1）substring\_index

[substring\_index](https://so.csdn.net/so/search?q=substring_index&spm=1001.2101.3001.7020)(‘s’，‘分隔符’，数字) #数字为正，从左往右有几个；数字为负，从右往左数几个



select if(substring\_index(profile, ',', -1)='male','male','female') gender,count(\*)

from user\_submit group by gender;

#### 6、窗口函数

<https://blog.csdn.net/lllmmm13/article/details/125766973?ops_request_misc=&request_id=&biz_id=102&utm_term=mysql%20%E7%AA%97%E5%8F%A3%E5%87%BD%E6%95%B0%20&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-9-125766973.nonecase&spm=1018.2226.3001.4187>





答案：

SELECT device\_id,university,gpa

FROM(SELECT device\_id,university,gpa,

      RANK() over (PARTITION BY university ORDER BY gpa) rk FROM user\_profile) as up

WHERE up.rk = 1;

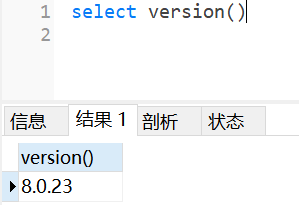
#### 7、其他函数

##### version()

查看数据库版本

**示例**

select version()



##### user()

查看数据库当前用户

**示例**

select user()

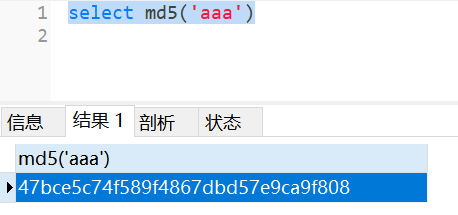


##### md5(expr)

返回参数md5加密后的值

**示例**

select md5('aaa')



### （四）聚合函数

聚合函数查询是纵向查询，它是对一列的值进行计算，然后返回一个结果值。

**聚合函数会忽略空值。**

五个聚合函数：

count： 统计指定列记录数，记录为NULL的不统计

sum： 计算指定列的数值和，如果不是数值类型，那么计算结果为0

max： 计算指定列的最大值

min： 计算指定列的最小值

avg： 计算指定列的平均值，如果不是数值类型，那么计算结果为0

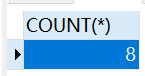
#### count函数

统计指定列记录数

**示例**

查询总条数

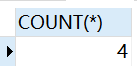
SELECT COUNT(\*) FROM student3;



**示例2**

查询年龄大于40的总人数

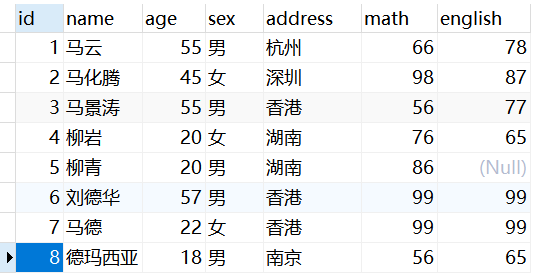
SELECT COUNT(\*) FROM student3 WHERE age>40;



说明：先查询后进行聚合函数统计

**示例3**

SELECT COUNT(english) FROM student3



**说明：忽略了null值**

**加上instinct可进行去重，例如，count(count math),统计数学成绩共有几个不同的分数**

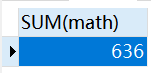
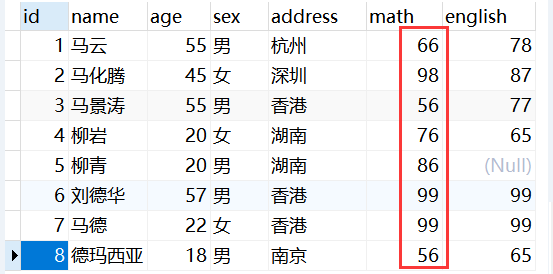
#### sum函数

计算指定列的数值和，如果不是数值类型，那么计算结果为0

示例

计算math的总和

SELECT SUM(math) FROM student3;



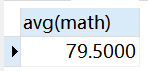
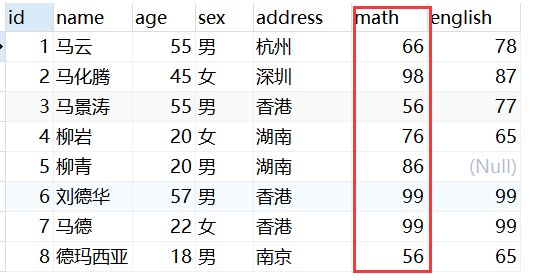
#### avg函数

计算指定列的平均值，如果不是数值类型，那么计算结果为0

**示例**

查询数学成绩平均分

SELECT AVG(math) FROM student3;



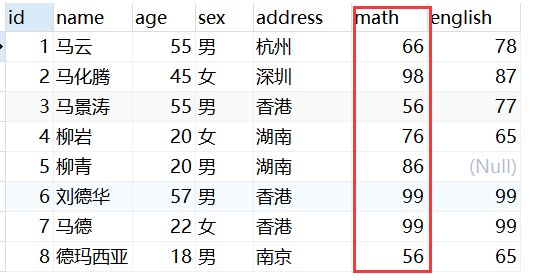
#### 4、max函数

计算指定列的最大值

**示例**

查询数学成绩最高分

SELECT MAX(math) FROM student3;



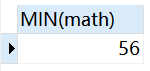
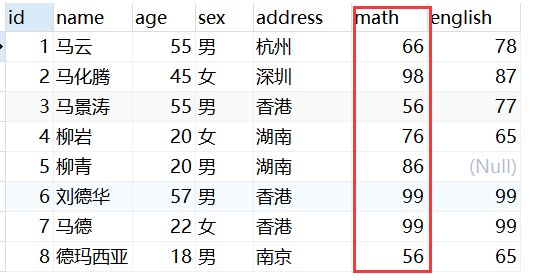
#### 5、min函数

计算指定列的最小值

**示例**

查询数学成绩最低分

SELECT MIN(math) FROM student3;



### （五）分组

分组查询是指使用 GROUP BY语句对查询信息进行分组，相同数据作为一组。

**语法**

SELECT 字段1,字段2... FROM 表名 GROUP BY 分组字段 [HAVING 条件];

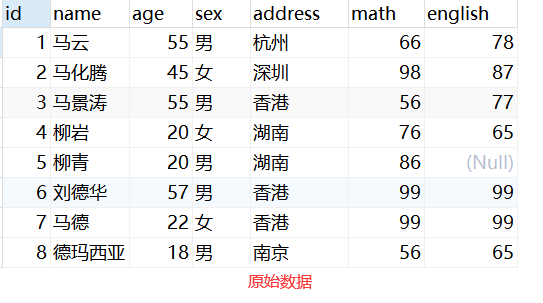
#### 1、分组查询(根据划分依据仅对查询结果进行分组)

GROUP BY怎么分组的？**将分组字段结果中相同内容作为一组**

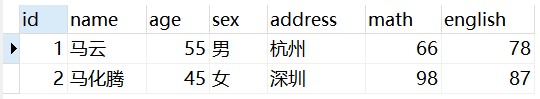
**示例1**

SELECT \* FROM student3 GROUP BY sex;

这句话会将sex相同的数据作为一组。



第一步：将sex相同的数据作为一组，分为男，女两组。



第二步：返回每组的第一条数据，作为查询结果。

**说明**

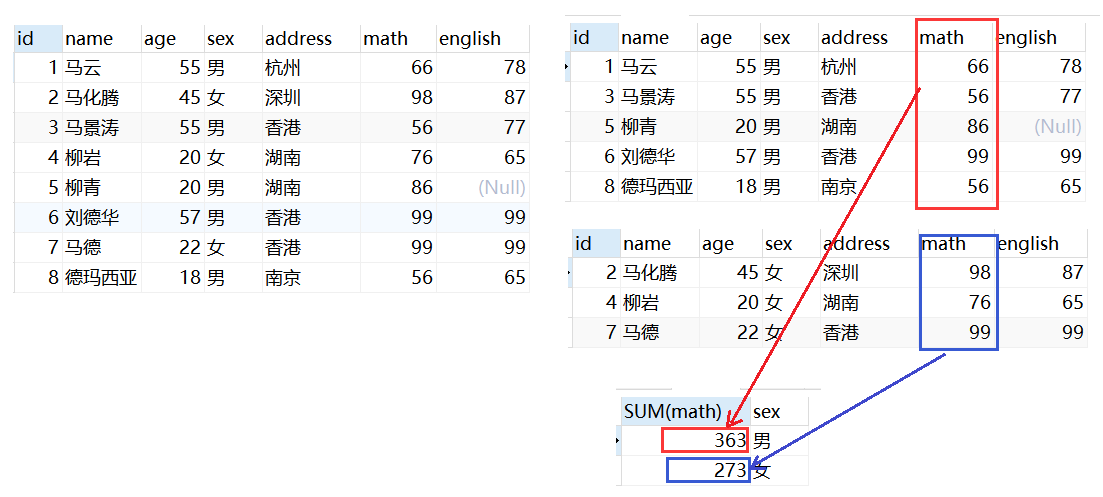
以上的操作是将分组字段结果中相同内容作为一组，并且返回每组的第一条数据。这样的操作显然没有什么意义。而且在有些数据库中（例如oracle）中，这样的语句是错误的！

**分组的目的就是为了统计，一般分组会跟聚合函数一起使用。**

**示例2**

SELECT SUM(math), sex FROM student3 GROUP BY sex;

是将每组的math进行求和，返回每组统计的结果。



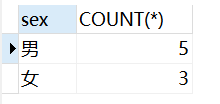
聚合函数是在分组后运行，统计每组的数据

**注意事项**

当我们使用某个字段分组，在查询的时候最好将这个字段查询出来，否则看不到数据属于哪组的。

**示例3**

查询男女各多少人  
SELECT sex, COUNT(\*) FROM student3 GROUP BY sex;

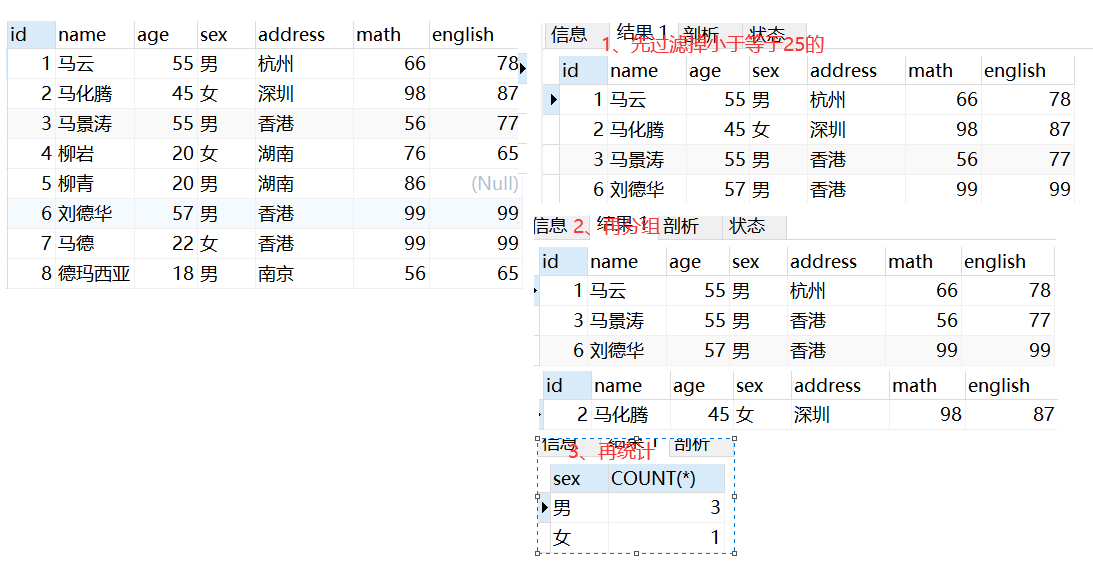


#### 2、分组时的条件过滤

**示例**

查询年龄大于25岁的人，按性别分组,统计每组的人数

先过滤掉年龄小于25岁的人；再分组；最后统计每组的人数  
SELECT sex, COUNT(\*) FROM student3 WHERE age > 25 GROUP BY sex;



**示例2**

查询年龄大于25岁的人，按性别分组，统计每组的人数，并只显示性别人数大于2的数据。

**错误示范**

SELECT sex, COUNT(\*) FROM student3 WHERE age > 25 GROUP BY sex WHERE COUNT(\*) >2;

**一个SQL语句中不应该出现两个where子句！**

**使用having子句解决**

SELECT sex, COUNT(\*) FROM student3 WHERE age > 25 GROUP BY sex HAVING COUNT(\*) >2;  
只有分组后人数大于2的`男`这组数据显示了出来。



#### 3、where和having的对比

1.having是在分组后对数据进行过滤，where是在分组前对数据进行过滤

2.having语句中可以使用聚合函数，where后面不可以使用聚合函数

### （六）limit

LIMIT是限制的意思，所以LIMIT的作用就是限制查询记录的条数。

**准备数据**

INSERT INTO student3(id,NAME,age,sex,address,math,english) VALUES   
(9,'唐僧',25,'男','长安',87,78),  
(10,'孙悟空',18,'男','花果山',100,66),  
(11,'猪八戒',22,'男','高老庄',58,78),  
(12,'沙僧',50,'男','流沙河',77,88),  
(13,'白骨精',22,'女','白虎岭',66,66),  
(14,'蜘蛛精',23,'女','盘丝洞',88,88);

**语法**

SELECT \*|字段列表 [as 别名] FROM 表名 [WHERE子句] [GROUP BY子句][HAVING子句][ORDER BY子句][LIMIT子句];

**注意：limit子句要放在SQL语句的最后**

**LIMIT语法格式**

LIMIT offset,length;或者limit length;

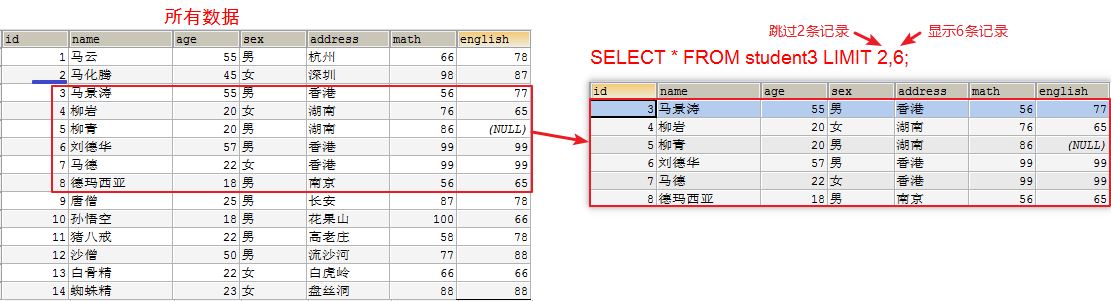
offset是指偏移量，可以认为是跳过的记录数量，默认为0，表示从哪开始

length是指需要显示的总记录数，表示显示几条

**示例**

查询学生表中数据，从第三条开始显示，显示6条。

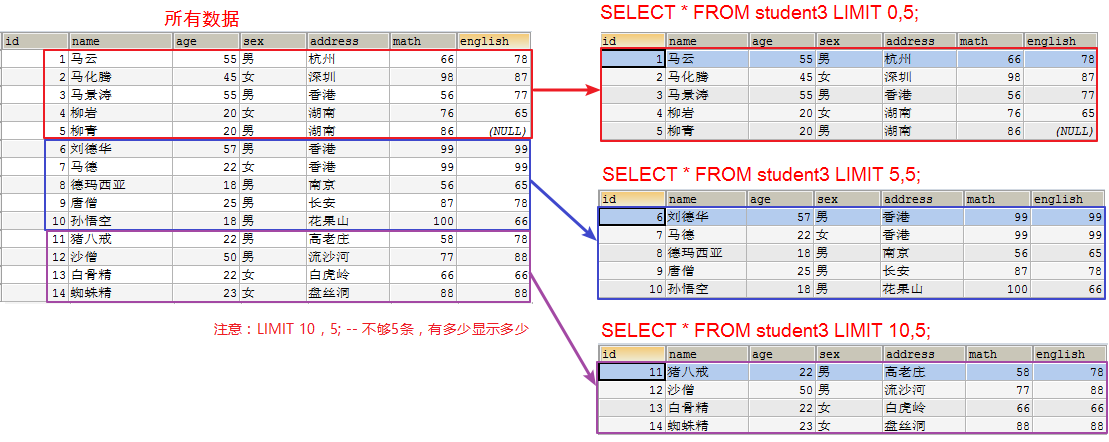
SELECT \* FROM student3 LIMIT 2,6;



**LIMIT的使用场景**：分页。比如我们登录京东、淘宝，返回的商品信息可能有几万条，不是一次全部显示出来，是一页显示固定的条数。 假设我们以每页显示5条记录的方式来分页。

**SQL语句如下**

-- 每页显示5条  
-- 第一页： LIMIT 0,5; 跳过0条，显示5条  
-- 第二页： LIMIT 5,5; 跳过5条，显示5条  
-- 第三页： LIMIT 10,5; 跳过10条，显示5条  
SELECT \* FROM student3 LIMIT 0,5;  
SELECT \* FROM student3 LIMIT 5,5;  
SELECT \* FROM student3 LIMIT 10,5;



**注意**

1. 如果第一个参数是0可以简写：

SELECT \* FROM student3 LIMIT 0,5;

SELECT \* FROM student3 LIMIT 5;

1. LIMIT 10,5; -- 如果不够5条，则有多少显示多少。

### （七）复合查询Union

在大多数开发中，使用一条SELECT查询就会返回一个结果集。如果，我们想一次性查询多条[SQL语句](https://so.csdn.net/so/search?q=SQL%E8%AF%AD%E5%8F%A5&spm=1001.2101.3001.7020" \t "_blank)，并将每一条SELECT查询的结果合并成一个结果集返回。就需要用到Union操作符，将多个SELECT语句组合起来，这种查询被称为并（Union）或者复合查询。

复合组合查询适用于下面两种情境中：

从多个表中查询出相似结构的数据，并且返回一个结果集

从单个表中多次SELECT查询，将结果合并成一个结果集返回，UNION ALL可以保留重复数据，不加ALL默认去掉重复数据。

如：

select user\_id,user\_nickname,user\_status from yy\_user where user\_status = 1

UNION

select user\_id,user\_nickname,user\_status from yy\_user where user\_id > 3

如：

select device\_id,gender,age,gpa from user\_profile where university='山东大学'

UNION ALL

select device\_id,gender,age,gpa from user\_profile where gender='male';

### （八）select语句总结

#### 1、书写顺序

**SELECT 字段 FROM 表名 [WHERE 条件] [GROUP BY 字段] [HAVING 条件] [ORDER BY 字段] [LIMIT offset,length];**

其中，SELECT字段，FROM表名为 必写，其余均可根据情况而定。

#### 2、执行顺序

1、from表名

2、where条件

3、group by 字段

4、having 条件

5、select 字段

6、order by 字段

7、limit

## 八、约束和策略

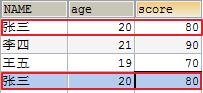
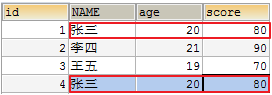
约束的作用：对表中的数据进行进一步的限制，保证数据的**正确性**、**有效性**和**完整性**。约束都是加在列上的，对一列的数据进行限制。

分类：数据库约束主要包括：主键约束、非空约束、唯一约束、默认约束(缺省约束)、外键约束、检查约束(MySQL不支持检查约束)

### 主键约束

#### 1、主键的作用

**用来唯一标识一条记录**，每个表都应该有一个主键，并且每个表只能有一个主键。 有些记录的 name,age,score 字段的值都一样时，那么就没法区分这些数据，造成数据库的记录不唯一，这样就不方便管理数据。

**哪个字段应该作为表的主键？**

通常不用业务字段作为主键，单独给每张表设计一个id的字段，把id作为主键。主键是给数据库和程序使用的，不是给最终的客户使用的。所以主键有没有含义没有关系，只要不重复，非空就行。

#### 2、主键的特点

主键列的特点：非空、唯一。

添加了主键约束的列，列中的值必须是**非空**（不能是null）和**唯一**的（各个行的数据都不相同）

#### 3、创建主键

##### 建表时添加主键

###### 方式一

**示例**

create table student(

id int **primary key**,

name varchar(20)

)

###### 方式二

**示例**

create table student(

id int ,

name varchar(20),

**primary key(id)**

)

##### 修改表方式添加主键

**示例**

create table stu1(

id int ,

name varchar(20),

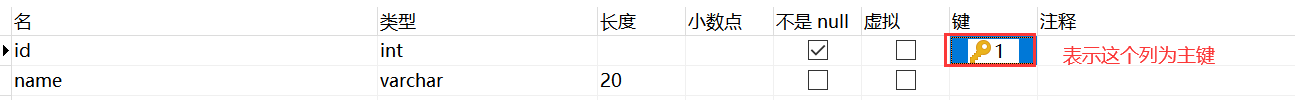
)

**alter table stu1 add CONSTRAINT stu1\_pk primary key(id)**

**stu1\_pk 约束名，可有可无**

##### Navicat中添加主键

建表时，在要作为主键的列处单击



#### 4、删除主键

**语法**

ALTER TABLE 表名 DROP PRIMARY KEY;

**示例**

删除student表的主键

ALTER TABLE stu1 DROP PRIMARY KEY;

#### 5、主键自增策略

主键如果让我们自己添加很有可能重复，我们通常希望在每次插入新记录时，数据库自动生成主键字段的值。可以在主键列上添加自增策略。

##### （1）建表时添加主键自增策略

**语法**

列名 数据类型 primary key auto\_increment

**自动增长字段类型必须是整数类型**

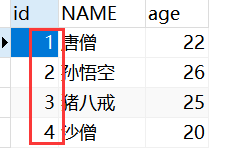
**示例**

创建学生表stu2, 包含字段(id, name, age)。将id做为主键并自动增长

CREATE TABLE stu2 (  
 id INT PRIMARY KEY **AUTO\_INCREMENT**,  
 NAME VARCHAR(20),  
 age INT  
 );

-- 插入数据

-- 主键默认从1开始自动增长  
INSERT INTO stu2(NAME, age) VALUES ('唐僧', 22);  
INSERT INTO stu2(NAME, age) VALUES ('孙悟空', 26);  
INSERT INTO stu2(NAME, age) VALUES ('猪八戒', 25);  
INSERT INTO stu2(NAME, age) VALUES ('沙僧', 20);



##### 修改表方式添加自增策略

**语法**

alter table 表名modify 主键列列名 数据类型 auto\_increment

**示例**

CREATE TABLE stu2 (  
 id INT PRIMARY KEY,  
 NAME VARCHAR(20),  
 age INT  
 );

**alter table stu2 modify id int auto\_increment**

##### Navicat中添加自增策略





##### （4）扩展

默认地AUTOINCREMENT 的开始值是1，如果希望修改起始值，请使用下列SQL

**语法**

ALTER TABLE 表名 AUTOINCREMENT=起始值;

#### 6、truncate截断表

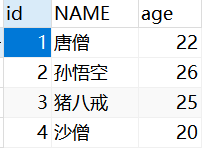
truncate表示截断表，会删除表中的全部数据，相当于删除表后重新创建该表。

**语法**

truncate table 表名;

**示例**

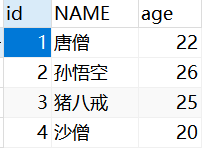
truncate table stu2



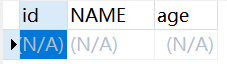
**DELETE和TRUNCATE对于主键自增策略的区别**

1. DELETE 删除表中的数据，但不重置AUTO\_INCREMENT的值。

**示例**



delete from stu2;



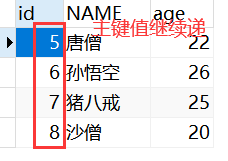
重新添加数据

INSERT INTO stu2(NAME, age) VALUES ('唐僧', 22);

INSERT INTO stu2(NAME, age) VALUES ('孙悟空', 26);

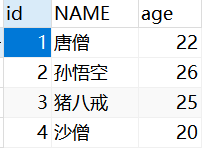
INSERT INTO stu2(NAME, age) VALUES ('猪八戒', 25);

INSERT INTO stu2(NAME, age) VALUES ('沙僧', 20);

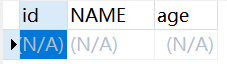


1. TRUNCATE 摧毁表，重建表，AUTO\_INCREMENT重置为1

**示例**



truncate table stu2;



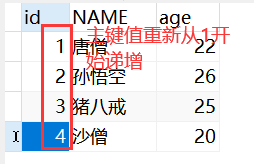
重新添加数据

INSERT INTO stu2(NAME, age) VALUES ('唐僧', 22);

INSERT INTO stu2(NAME, age) VALUES ('孙悟空', 26);

INSERT INTO stu2(NAME, age) VALUES ('猪八戒', 25);

INSERT INTO stu2(NAME, age) VALUES ('沙僧', 20);



### 非空约束

添加非空约束的字段表示这个字段的每一行的值都不能为null值。

#### 1、建表时添加非空约束

**语法**

字段名 字段类型 NOT NULL;

**示例**

创建表学生表stu3，包含字段(id,name,gender) ，其中name不能为NULL

CREATE TABLE stu3 (  
 id int,  
 name varchar(20) NOT NULL,  
 gender char(1)  
 );

INSERT INTO st8 VALUES (1, '张三', '男');  
INSERT INTO st8 VALUES (2, '李四', '男');  
INSERT INTO st8 VALUES (3, '王五', '男');  
INSERT INTO st8 VALUES (4, '赵六', '男');  
  
-- **姓名不赋值出现姓名不能为null: Column 'name' cannot be null**  
INSERT INTO st8 VALUES (5, NULL, '男');

#### 2、修改表的方式添加非空约束

**语法**

alter table 表名 modify 列名 数据类型 not null;

**示例**

CREATE TABLE stu3 (  
 id int,  
 name varchar(20) NOT NULL,  
 gender char(1)  
 );

**alter table stu3 modify name varchar(20) not null**

### （三）唯一约束

添加了唯一约束的列的值不能重复，但是可以为null。

#### 创建唯一约束

##### 建表时添加唯一约束

**语法**

列名 数据类型 UNIQUE

**示例**

创建stu4表，包含字段(id, name,email) ，email这一列设置唯一约束，不能重复。

CREATE TABLE stu4 (  
 id INT primary key,  
 NAME VARCHAR(20),

email varchar(200) unique  
)

insert into stu4 values(1,'张三','zs@aaa.com');

insert into stu4 values(2,'李四','ls@aaa.com');

insert into stu4 [values(3,'王五','ww@aaa.com');](mailto:values(3,'王五','ww@aaa.com');)

-- Duplicate entry 'zs@aaa.com' for key 'stu4.email' email 重复

insert into stu4 [values(4,'赵六','zs@aaa.com');](mailto:values(4,'赵六','zs@aaa.com');)

##### 修改表方式添加唯一约束

**语法**

alter table 表名 add constraint [约束名] unique(字段名)

**示例**

create table stu4(

id int primary key auto\_increment,

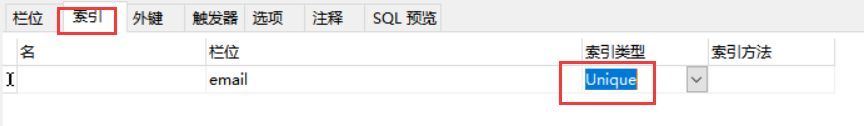
name varchar(20),

email varchar(200)

)

**alter table student add constraint stu4\_unq unique(email)**  stu4\_unq是约束名，可以省略

##### Navicat中添加唯一约束



#### 删除唯一约束

**语法**

alter table 表名 drop index 唯一约束的约束名

**示例**

alter table student drop index stu4\_unq

### （四）缺省（默认）约束

缺省约束是指指定某个列的缺省值，当该列在插入数据时不指定值，将会使用缺省值。

#### 1、创建缺省约束

##### （1）建表时添加缺省约束

**语法**

列名 数据类型 default ‘缺省值’

**示例**

CREATE TABLE stu5 (

id INT primary key,

NAME VARCHAR(20),

address varchar(200) default '北京'

)

insert into stu5(id,name) values(1,'张三');

insert into stu5(id,name,address) values(2,'李四','上海');

select \* from stu5;



##### （2）修改表的方式添加缺省约束

**语法**

alter table 表名 change column 列名 列名 数据类型 default ‘缺省值’

**示例**

CREATE TABLE stu5 (

id INT primary key,

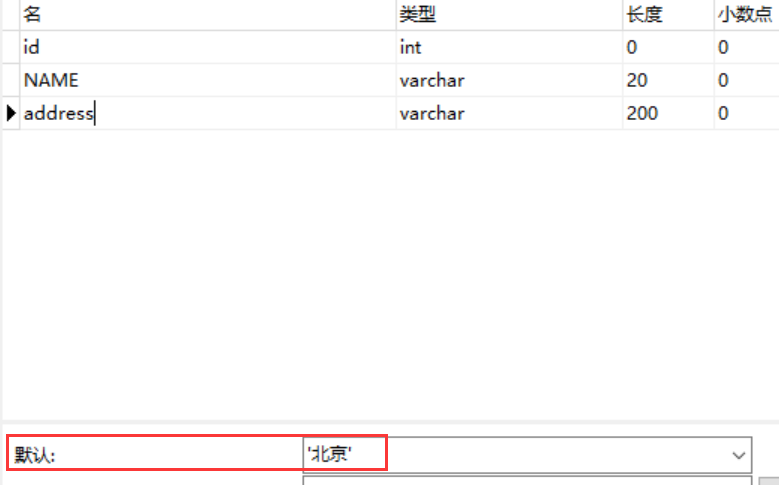
NAME VARCHAR(20),

address varchar(200)

)

alter table stu5 change column address address varchar(200) default '北京';

##### （3）Navicat中添加缺省约束



#### 2、删除缺省约束

**语法**

alter table 表名 change column 列名 列名 数据类型 default null

**示例**

alter table stu5 change column address adress varchar(200) default null

### （五）外键约束

#### 1、单表的缺点

创建一个员工表，包含如下列(id, name, age, dep*name, dep*location) ，id主键并自动增长，添加6条数据

CREATE TABLE emp (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(30),  
 age INT,  
 dep\_name VARCHAR(30),  
 dep\_location VARCHAR(30)  
);  
  
-- 添加数据  
INSERT INTO emp (NAME, age, dep\_name, dep\_location) VALUES ('张三', 20, '研发部', '广州');  
INSERT INTO emp (NAME, age, dep\_name, dep\_location) VALUES ('李四', 21, '研发部', '广州');  
INSERT INTO emp (NAME, age, dep\_name, dep\_location) VALUES ('王五', 20, '研发部', '广州');  
  
INSERT INTO emp (NAME, age, dep\_name, dep\_location) VALUES ('老王', 20, '销售部', '深圳');  
INSERT INTO emp (NAME, age, dep\_name, dep\_location) VALUES ('大王', 22, '销售部', '深圳');  
INSERT INTO emp (NAME, age, dep\_name, dep\_location) VALUES ('小王', 18, '销售部', '深圳');

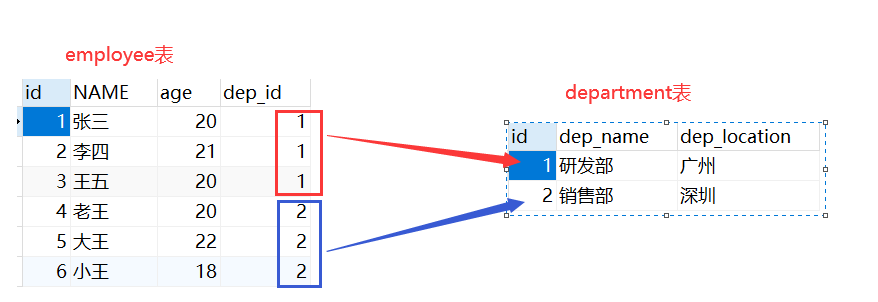
**缺点:表中出现了很多重复的数据(数据冗余)，如果要修改研发部的地址需要修改3个地方。**



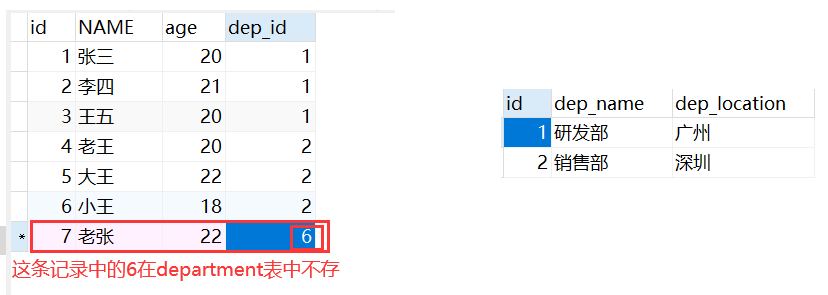
#### 2、解决方案

将一张表分成2张表(员工表和部门表)

-- 创建部门表  
CREATE TABLE department (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 dep\_name VARCHAR(20),  
 dep\_location VARCHAR(20)  
);  
  
-- 创建员工表  
CREATE TABLE employee (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(20),  
 age INT,  
 dep\_id INT  
);  
  
-- 添加2个部门  
INSERT INTO department (dep\_name, dep\_location) VALUES ('研发部', '广州'), ('销售部', '深圳');  
  
-- 添加员工,dep\_id表示员工所在的部门  
INSERT INTO employee (NAME, age, dep\_id) VALUES   
('张三', 20, 1),   
('李四', 21, 1),   
('王五', 20, 1),   
('老王', 20, 2),  
('大王', 22, 2),  
('小王', 18, 2);



**问题**： 当我们在employee的dep*id里面输入不存在的部门,数据依然可以添加.但是并没有对应的部门，不能出现这种情况。employee的dep*id中的内容只能是department表中存在的id



**目标：**需要约束dep\_id只能是department表中已经存在id

**解决方式：使用外键约束。**

#### 3、什么是外键约束

一张表中的某个字段引用另一个表的主键。

主表： 约束别人的表

副表/从表： 使用别人的数据，被约束的表



#### 4、创建外键

##### （1）建表时创建外键约束

**语法**

[CONSTRAINT] [外键约束名称] FOREIGN KEY(外键字段名) REFERENCES 主表名(主键字段名)

关键字解释：

CONSTRAINT：约束关键字

FOREIGN KEY(外键字段名)：某个字段作为外键

REFERENCES 主表名(主键字段名)：表示参照主表中的某个字段

**要求：外键列的数据类型和长度必须和被参照的主键列的数据类型和长度一致**

**示例**

在创建副表/从表之前要先创建主表

CREATE TABLE department (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 dep\_name VARCHAR(20),  
 dep\_location VARCHAR(20)  
 );

副表/从表: 被别的表约束，表结构添加外键约束

CREATE TABLE employee (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(20),  
 age INT,  
 dep\_id INT,  
 -- 添加一个外键  
 -- 外键取名公司要求,一般fk结尾  
 **CONSTRAINT emp\_depid\_ref\_dep\_id\_fk FOREIGN KEY(dep\_id) REFERENCES department(id)**  
);

正常添加数据

INSERT INTO employee (NAME, age, dep\_id) VALUES  
('张三', 20, 1),  
('李四', 21, 1),  
('王五', 20, 1),  
('老王', 20, 2),  
('大王', 22, 2),  
('小王', 18, 2);

部门错误的数据添加失败

INSERT INTO employee (NAME, age, dep\_id) VALUES ('二王', 20, 5);

##### 修改表的方式添加外键

**语法**

alter table 从表名 add constraint [约束名] foreign key(外键列) reference 主表(主表主键列)

**示例**

CREATE TABLE employee (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(20),  
 age INT,  
 dep\_id INT  
 );

alter table employee add constraint emp\_fk1 foreign key(dept\_id) reference department(id)

#### 5、外键的级联（重要功能）

##### （1）问题

**要把部门表中的id值2，改成5，能不能直接修改呢？**

UPDATE department SET id=5 WHERE id=2;

不能直接修改：Cannot delete or update a parent row: a foreign key constraint fails

如果副表(员工表)中有引用的数据，不能直接修改主表(部门表)主键。

**要删除部门id等于1的部门，能不能直接删除呢？**

DELETE FROM department WHERE id=1;

不能直接删除：Cannot delete or update a parent row: a foreign key constraint fails

如果副表(员工表)中有引用的数据，不能直接删除主表(部门表)数据。

##### （2）级联操作

在修改和删除主表的主键时，同时更新或删除副表的外键值，称为级联操作。

**语法**

ON UPDATE CASCADE -- 级联更新，主键发生更新时，外键也会更新

ON DELETE CASCADE -- 级联删除，主键发生删除时，外键也会删除

**示例**

删除employee表

重新创建employee表，添加级联更新和级联删除

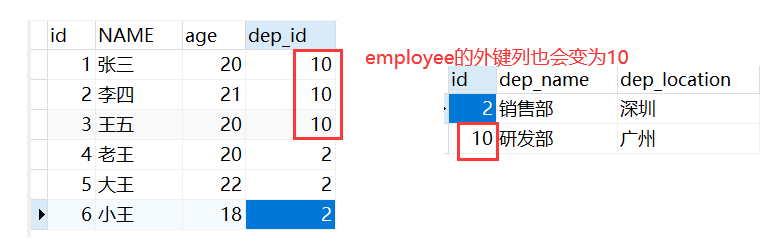
CREATE TABLE employee (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(30),  
 age INT,  
 dep\_id INT,  
 -- 添加外键约束,并且在从表中添加级联更新和级联删除  
 CONSTRAINT employee\_dep\_fk FOREIGN KEY (dep\_id) REFERENCES department(id) **ON UPDATE CASCADE ON DELETE CASCADE**);

再次添加数据到员工表和部门表

INSERT INTO employee (NAME, age, dep\_id) VALUES ('张三', 20, 1);  
INSERT INTO employee (NAME, age, dep\_id) VALUES ('李四', 21, 1);  
INSERT INTO employee (NAME, age, dep\_id) VALUES ('王五', 20, 1);  
INSERT INTO employee (NAME, age, dep\_id) VALUES ('老王', 20, 2);  
INSERT INTO employee (NAME, age, dep\_id) VALUES ('大王', 22, 2);  
INSERT INTO employee (NAME, age, dep\_id) VALUES ('小王', 18, 2);

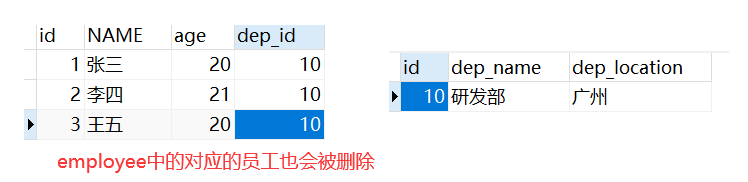
**把部门表中id等于1的部门改成id等于10**

UPDATE department SET id=10 WHERE id=1;



**删除部门号是2的部门**

DELETE FROM department WHERE id=2



**注意：要慎用级联操作**

## 九、表关系

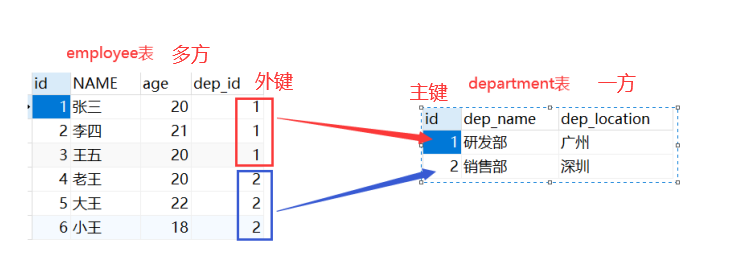
### （一）概念和分类

现实生活中，实体与实体之间是存在关系的，比如：老公和老婆，部门和员工，老师和学生等。我们设计的表是对现实中实体的描述，那么我们在设计表的时候，就应该体现出表与表之间的这种关系。

表与表之间的关系主要包括：一对多（多对一）、一对一、多对多三种。

### （二）一对多（多对一）

一对多（1:n）。 例如：班级和学生，部门和员工，客户和订单，分类和商品等

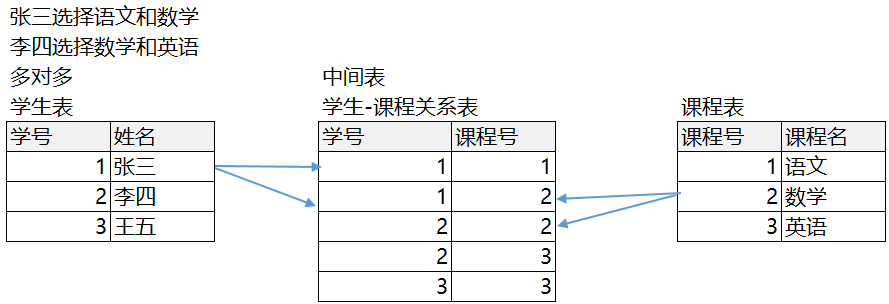
一对多建表原则: 在从表(多方)创建一个字段，字段作为外键指向主表(一方)的主键 

建表操作和讲解外键时相同

### 多对多

多对多（m:n）。例如：老师和学生，学生和课程，用户和角色。

多对多关系建表原则: 需要创建第三张表，中间表中至少两个字段， 。



**示例**

**创建学生表student**

create table student(

id int primary key auto\_increment,

name varchar(20)

)

**创建课程表course**

create table course(

id int primary key auto\_increment,

name varchar(20)

)

**创建中间关系表 student\_course,中间关系表中student\_id和course\_id联合做主键，分别做外键参照多方的两端**

create table student\_course(

student\_id int,

course\_id int,

**primary key(student\_id,course\_id),**

constraint foreign key(student\_id) references student(id),

constraint foreign key(course\_id) references course(id)

)

### 一对一

一对一（1:1）。 在实际应用中并不多见，因为一对一可以创建成一张表。

#### 主键关联方式(本质上是一对多，主键对外键，外键并不唯一)

主表的主键和从表的主键，形成主外键关系。

**示例**

**创建丈夫表husband**

create table husband(

id int primary key auto\_increment,

name varchar(20)

)

**创建妻子表wife，wife的主键列id作为外键参照husband的主键列id**

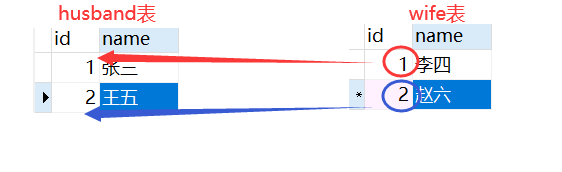
create table wife(

id int primary key,

name varchar(20),

constraint foreign key(id) references husband(id)

)



#### 唯一外键方式

主表的主键和从表的外键（唯一），形成主外键关系。

**示例**

创建person表

create table person(

id int primary key auto\_increment,

name varchar(20)

);

创建idcard表，idcard表中的person\_id做外键参照person表的id，在person\_id列上添加唯一约束，就表示出了person和idcard的一对一关系。

create table idcard(

id int primary key,

card\_num varchar(20),

person\_id int **unique**,

constraint foreign key(person\_id) references person(id)

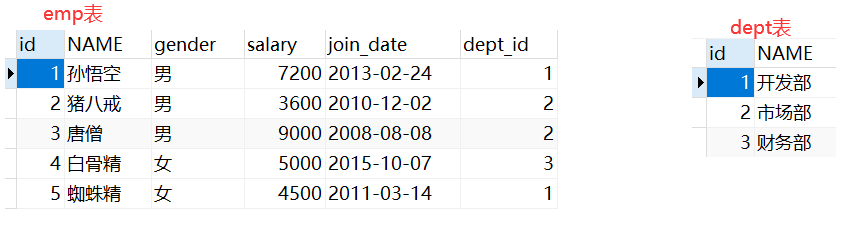
)



## 十、多表查询

### 什么是多表查询

之前我的查询都是从一张表中返回数据。同时查询多张表获取到需要的数据，就是多表查询。比如：要查询员工的姓名和所在的部门名称，需要从employee表和department表两个表中查询数据。



**准备数据**

**创建部门表 dept**  
CREATE TABLE dept (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(20)  
);  
**添加数据**  
INSERT INTO dept (NAME) VALUES ('开发部'),('市场部'),('财务部');  
  
**创建员工表**  
CREATE TABLE emp (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(10),  
 gender CHAR(1), -- 性别  
 salary DOUBLE, -- 工资  
 join\_date DATE, -- 入职日期  
 dept\_id INT  
);

**添加数据**  
INSERT INTO emp(NAME,gender,salary,join\_date,dept\_id) VALUES('孙悟空','男',7200,'2013-02-24',1);  
INSERT INTO emp(NAME,gender,salary,join\_date,dept\_id) VALUES('猪八戒','男',3600,'2010-12-02',2);  
INSERT INTO emp(NAME,gender,salary,join\_date,dept\_id) VALUES('唐僧','男',9000,'2008-08-08',2);  
INSERT INTO emp(NAME,gender,salary,join\_date,dept\_id) VALUES('白骨精','女',5000,'2015-10-07',3);  
INSERT INTO emp(NAME,gender,salary,join\_date,dept\_id) VALUES('蜘蛛精','女',4500,'2011-03-14',1);

### （二）笛卡尔积

#### 1、什么是笛卡尔积现象

多表查询时左表的每条数据和右表的每条数据组合，这种效果成为笛卡尔积。

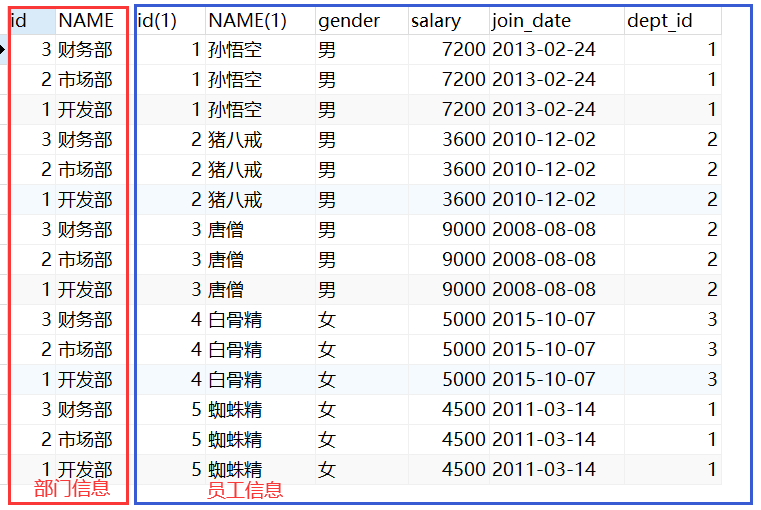
**语法**

select \* | 列1,列2,...,列n from 表1,表2,...,表n

**示例**

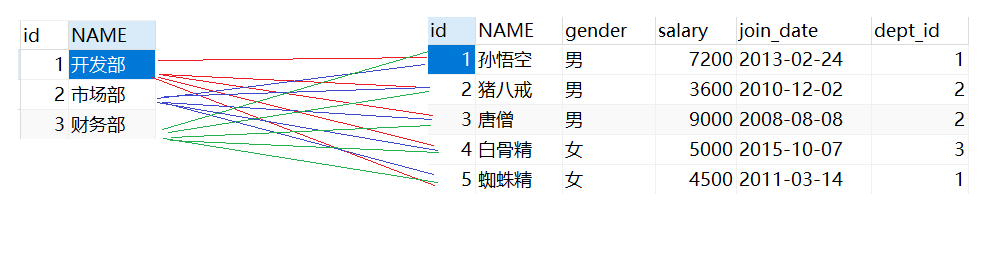
需求：查询部门和员工信息

SELECT \* FROM dept, emp;



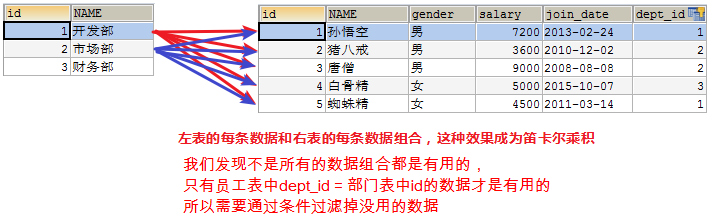
以上数据其实是左表的每条数据和右表的每条数据组合。左表有3条，右表有5条，最终组合后3\*5=15条数据。

**左表的每条数据和右表的每条数据组合，这种效果称为笛卡尔乘积，笛卡尔积中有很多没有意义的数据。**

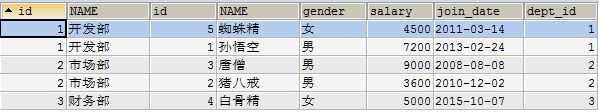


#### 2、如何清除笛卡尔积现象的影响

我们发现不是所有的数据组合都是有用的，只有员工表.dept\_id = 部门表.id 的数据才是有用的。所以需要通过条件过滤掉没用的数据。

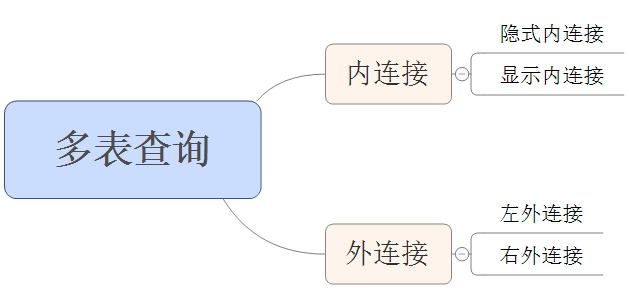


SELECT \* FROM dept, emp WHERE emp.`dept\_id`=dept.`id`;



### （三）连接查询

要清除笛卡尔积现象，可以使用连接查询。连接查询分为内连接和外连接。



#### 1、内连接（匹配不到的不保留）

用左边表的记录去匹配右边表的记录，符合条件的才显示。内连接又分为隐式内连接和显示内连接两种**。**

##### （1）隐式内连接

隐式内连接：不使用JOIN关键字，条件使用WHERE指定。

**语法**

SELECT 字段名 FROM 左表, 右表 WHERE 条件;

**示例**

1、查询员工的姓名和所在的部门的名称

select emp.name emp\_name,dept.name dept\_name from emp,dept where emp.dept\_id = dept.id

可以使用表别名的方式简化查询语句

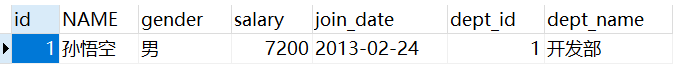
select e.name emp\_name,d.name dept\_name from emp e,dept d where e.dept\_id = d.id



**where 条件中的 e.dept\_id = d.id 就是连接条件**

2、查询员工孙悟空的信息和他所在的部门名称

select e.\*,d.name dept\_name from emp e,dept d where e.dept\_id=d.id and e.name='孙悟空'



##### （2）显示内连接

显示内连接：使用INNER JOIN ... ON语句，可以省略INNER。查询的效果和隐式内连接相同。

**语法**

SELECT 字段名 FROM 左表 INNER JOIN 右表 ON 条件；

**示例**

查询唐僧的信息，显示员工id、姓名、性别、工资和所在的部门名称，我们发现需要联合两张表才能同时查询出需要的数据，我们使用内连接。

select e.id emp\_id,e.name emp\_name,e.gender emp\_gender,e.salary emp\_salary,d.name dept\_name from emp e **inner join dept d on e.dept\_id=d.id** and e.name='唐僧'

但是显式连接可以减少字段的扫描，有更快的执行速度。

#### 2、外连接（匹配不到的也会保留，比如NULL）

##### （1）左外连接

左外连接：使用LEFT OUTER JOIN ... ON，OUTER可以省略。

左表数据全部显示，右表显示和左表有关联的数据。

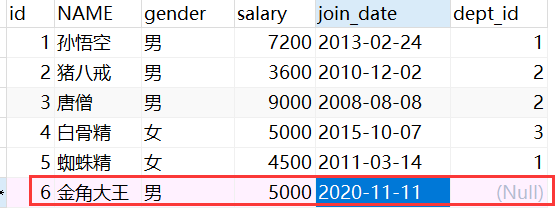
**语法**

SELECT 字段名 FROM 左表 LEFT OUTER JOIN 右表 ON 条件

用左边表的记录去匹配右边表的记录，如果符合条件的则显示；否则，显示NULL。

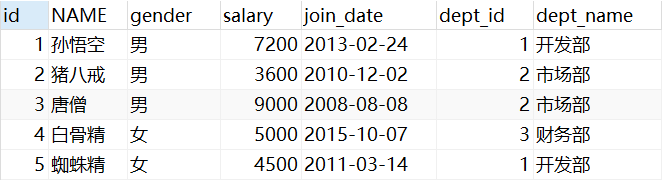
**示例**

新入职了一名员工‘金角大王’，但该员工没有分配部门。

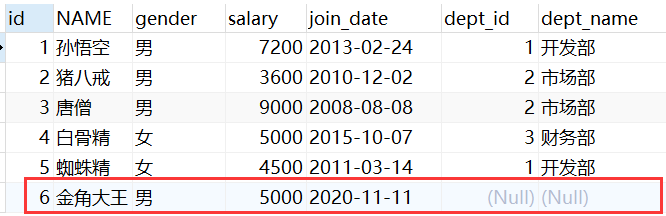


现在要查询员工的信息和所在部门的名称。如果使用内连接查询，没有部门的员工将不会显示，因为不满足连接条件。

select e.\*,d.name dept\_name from emp e,dept d where e.dept\_id=d.id



想要显示全部员工，可以把员工emp 当成左表，部门dept表当成右表，使用左外连接查询



##### （2）右外连接

右外连接：使用RIGHT OUTER JOIN ... ON，OUTER可以省略。右表数据全部显示，左表显示和右表有关联的数据。

**语法**

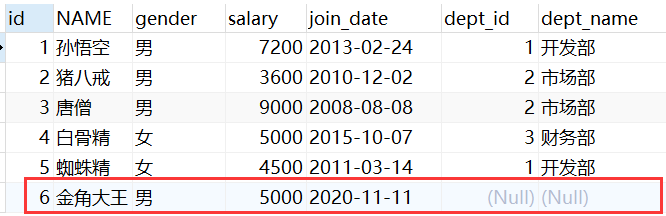
SELECT 字段名 FROM 左表 RIGHT OUTER JOIN 右表 ON 条件；

用右边表的记录去匹配左边表的记录，如果符合条件的则显示；否则，显示NULL。

**示例1**

使用右外连接解决上面左外连接解决的问题

select e.\*,d.name dept\_name from dept d right outer join emp e on e.dept\_id = d.id;



使用右外连接，把原来的左右表位置互换，产生同样的查询效果。

**三个表进行连接，最详细的放在前面作为左连接，其他两个进行LEFT JOIN，ON条件紧跟其后**

select university,difficult\_level,round(count(q1.question\_id)/count(distinct q1.device\_id),4) avg\_answer\_cnt

from question\_practice\_detail q1

LEFT JOIN user\_profile u on u.device\_id = q1.device\_id

LEFT JOIN question\_detail q2 on q1.question\_id = q2.question\_id

group by university,difficult\_level;

### （四）子查询

一条SELECT语句结果作为另一条SELECT语法的一部分。

**语法**

select 字段，... from 表 where 字段 运算符 (select 字段,... from 表)

子查询 (内查询) 在主查询之前执行一次。

子查询的结果被用于主查询 (外查询)

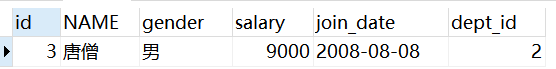
**示例**

查询出工资最高的员工信息

SELECT \* FROM employee WHERE salary=(SELECT MAX(salary) FROM employee);

子查询先查询出最高工资，主查询查询出拥有最高工资的员工信息





#### 1、子查询的使用原则

1、子查询放在圆括号中。

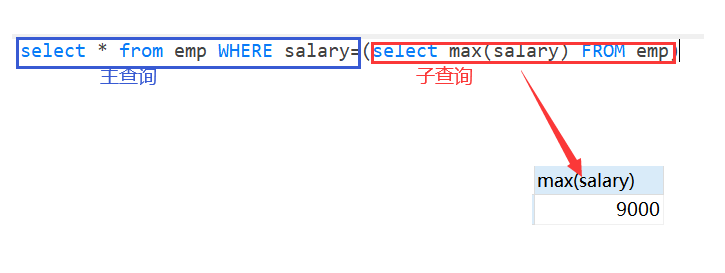
2、将子查询放在比较条件的右边。（非强制要求）

3、在子查询中的 ORDER BY 子句不需要。

4、在单行子查询中用单行运算符，在多行子查询中用多行运算符。

#### 2、单行子查询 (子查询是单例单行)

子查询的结果是一个值



**单行运算符**

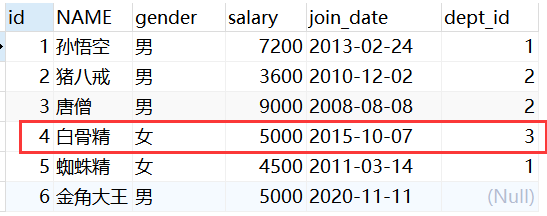
= > >= < <= != 等我们学习过的众多运算符都是单行运算符

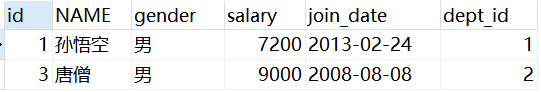
**示例**

查询工资比白骨精高的员工信息

select \* from emp where salary>(select salary from emp where name='白骨精')

子查询先查询出'白骨精'的工资，作为条件交给主查询使用





#### 3、多行子查询（子查询是单例多行）

子查询结果是单例多行

**多行运算符**

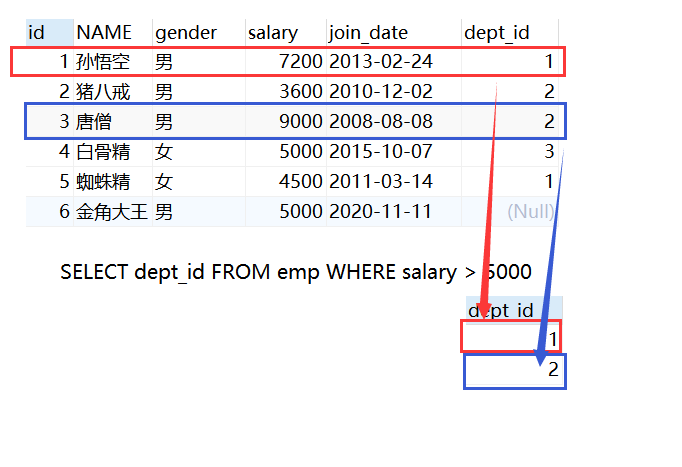
in、any、all

**示例**

**1、查询工资大于5000的员工，来自于哪些部门的名字**

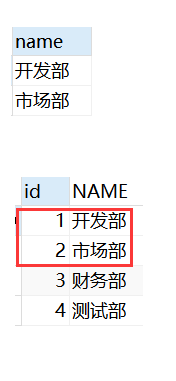
先查询大于5000的员工所在的部门id

SELECT dept\_id FROM emp WHERE salary > 5000



再查询在这些部门id中部门的名字

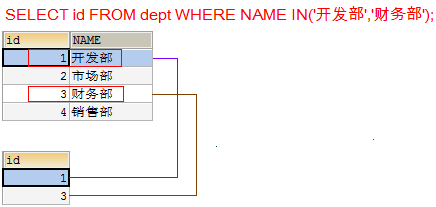
SELECT dept.name FROM dept WHERE dept.id IN (SELECT dept\_id FROM emp WHERE salary > 5000)



**2、查询开发部与财务部所有的员工信息**

先查询开发部与财务部的id

SELECT id FROM dept WHERE NAME IN('开发部','财务部');



再查询在这些部门id中有哪些员工

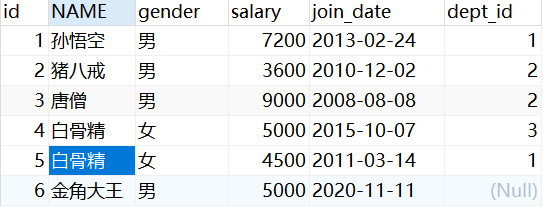
SELECT \* FROM emp WHERE dept\_id IN (SELECT id FROM dept WHERE NAME IN('开发部','财务部'))



any 和 all 不是单独使用的运算符，要和比较运算符一起使用

**示例**

修改表中数据，把蜘蛛精的name修改为白骨精

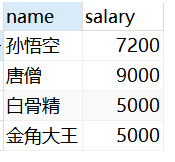


select salary from emp where name='白骨精'



**any表示任意值，满足一个就可以**

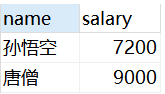
select name,salary from emp where salary >any(select salary from emp where name='白骨精')



>any表示比最小的大就可以(大于4500)

**all表示全部值，必须满足所有情况**

select name,salary from emp where salary >all(select salary from emp where name='白骨精')



>all 表示比最大的还要大(大于5000)

#### 4、多列子查询 （相当于是把子查询的结果作为了一个新表， 再次基础上做其他查询操作）

子查询结果是单列，在WHERE后面作为条件子查询结果，如上述子查询👆

**子查询结果是多列，肯定在FROM后面作为表，如下述子查询👇**

**语法**

SELECT 查询字段 FROM （子查询） 表别名 WHERE 条件

子查询结果只要是多行多列，肯定在FROM后面作为表 。

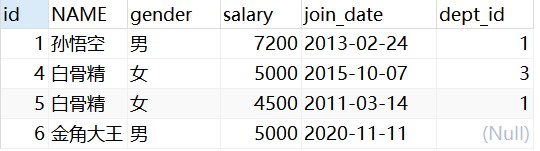
**子查询作为表需要取别名，否则这张表无法访问表中的字段。**

**示例**

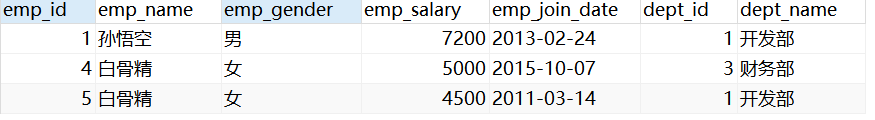
**查询出2011年以后入职的员工信息，包括部门名称**

在员工表中查询2011-1-1以后入职的员工

SELECT \* FROM emp WHERE join\_date > '2011-1-1'



查询所有的部门信息，与上面的虚拟表中的信息组合，找出所有部门id等于的dept\_id

SELECT e.id emp\_id,e.name emp\_name,e.gender emp\_gender,e.salary emp\_salary,e.join\_date emp\_join\_date,d.id dept\_id,d.name dept\_name FROM dept d, (SELECT \* FROM emp WHERE join\_date > '2011-1-1') e WHERE e.dept\_id = d.id

## 十一、数据库备份与还原

### （一）备份的应用场景

我们在服务器上进行数据传输、数据存储和数据交换，就有可能产生数据故障。比如发生意外停机或存储介质损坏。这时，如果没有采取数据备份和数据恢复手段与措施，就会导致数据的丢失，造成的损失是无法弥补与估量的。

### 备份

#### 命令方式备份

**语法**

mysqldump -u用户名 -p密码 数据库 > 文件的路径/文件名.sql

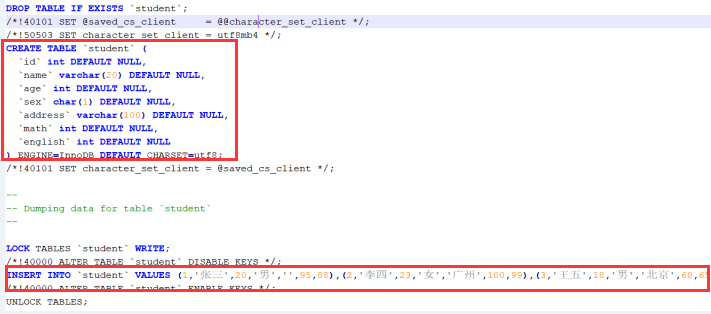
**示例**

备份mydb1数据库中的数据

mysqldump -uroot -pmysql mydb1 > D:/abc.sql

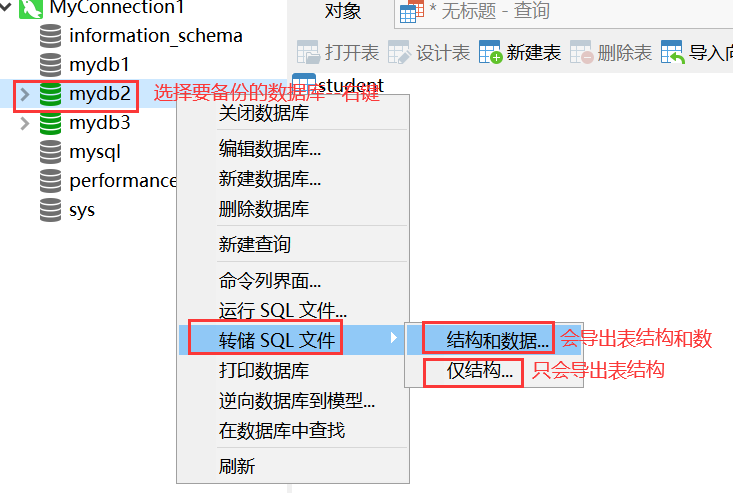






sql文件中将会包含相应的建表语句和插入语句

#### 使用Navicat备份



再选择路径保存相应的sql文件。

### 还原

#### 命令方式还原（先创建空数据库，使用数据库，再导入之前的sql文件）

**语法**

SOURCE 导入文件的路径;

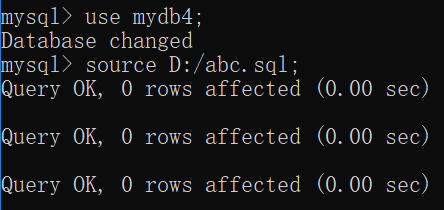
**示例**

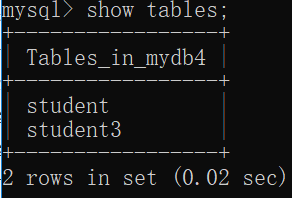
注意：还原时要先登录到MySQL中

创建一个新数据库mydb4，把前面导出的mydb1中的数据导入到mydb4中。

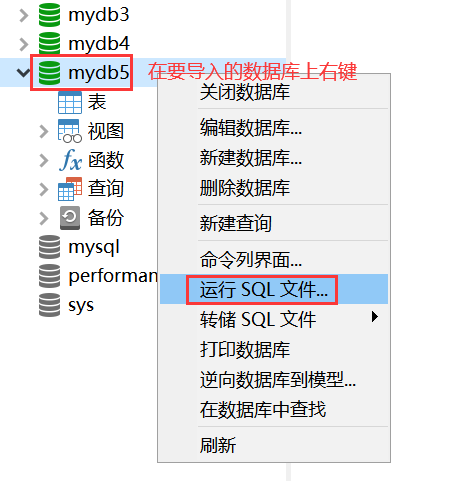
use mydb4;

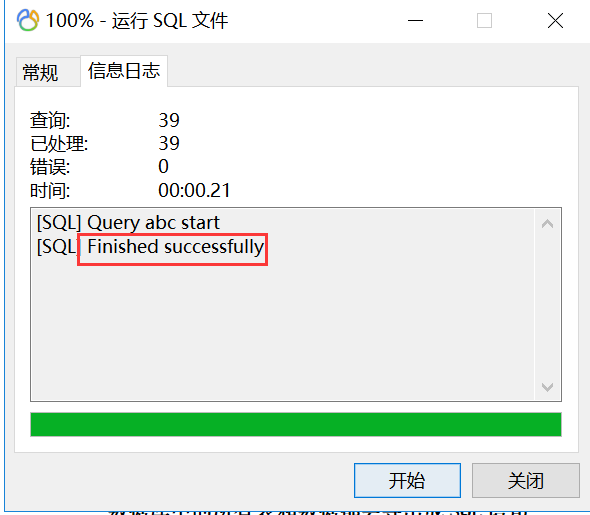
source D:/abc.sql；

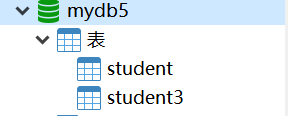




#### 使用Navicat还原







## 十二、数据库事务

### （一）什么是数据库事务

数据库事务是访问并可能操作各种数据项的一个数据库操作（包括读、写）序列，这些操作要么全部执行,要么全部不执行，是一个不可分割的工作单位。事务由事务开始与事务结束之间执行的全部数据库操作组成。

例如：银行转账。核心操作包括更新一个账户减去转账金额，更新另一个账户加上转账金额，要求要同时成功或同时失败。

### （二）事务的目的

1. 为数据库操作提供了一个从失败中恢复到正常状态的方法，同时提供了数据库即使在异常状态下仍能保持一致性的方法。
2. 当多个应用程序在并发方法数据库时，可以再这些应用程序之间提供一个隔离方法，以防止彼此的操作互相干扰。

总结：事务的目的就是要提供3种方法：1、失败恢复方法。 2、保持一致性的方法。3、操作隔离的方法。

**成功情况下**

能将数据从一种状态变为另一种状态，并能够持久化。

**异常情况下**

1. 能将数据恢复到正常状态。
2. 要能保证一致性，包含数据的一致性和约束的一致性。

**并发情况下**

并发的操作之间不能产生相互影响。

### 事务特性

事务包括ACID四个特性，分别是原子性、一致性、隔离性、持久性。

#### 原子性（Actomicity）

表示一个事务内的所有操作是一个整体，是一个不可分割的工作单位，要么全部成功，要么全部失败。

#### 一致性（Consistency）

事务前后数据的完整性必须保持一致。

#### 隔离性（Isolation）

事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。

#### 持久性（Durability）

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，接下来即使数据库发生故障也不应该对其有任何影响。

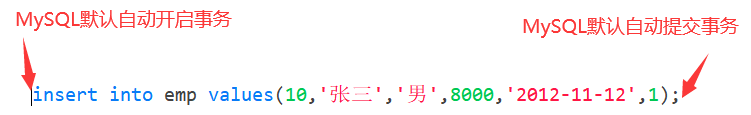
### 操作事务

在数据库中，一个事务可以由多条SQL语句组成，当然一个事务中也可以只包含一条SQL语句。

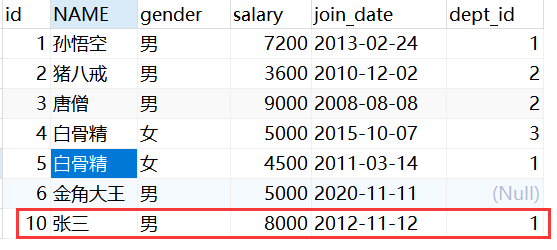
MYSQL中可以有两种方式进行事务的操作：1、自动提交事务。2、手动提交事务。

#### 自动提交事务

MySQL的每一条DML(增删改)语句都是一个单独的事务，MySQL默认情况下，在执行每条语句时都会**自动开启**一个事务，执行完毕**自动提交事务**。



当语句执行完成后，数据库中添加了数据。



#### 手动提交事务

事务有关的SQL语句

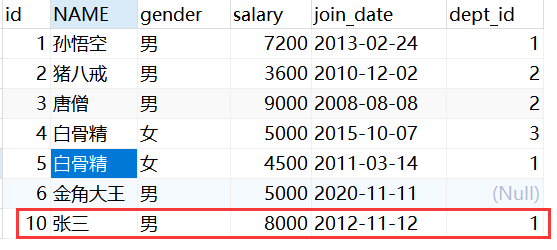
| SQL语句 | 描述 |
| --- | --- |
| start transaction; | 开启事务 |
| commit; | 提交事务 |
| rollback; | 回滚事务 |

**示例**

start transaction; --开启事务

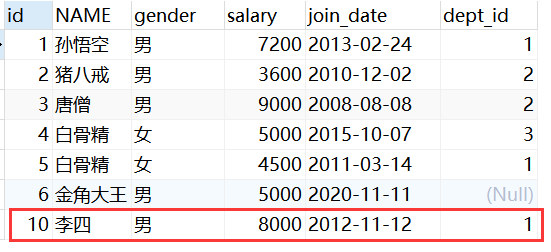
update emp set name='李四' where id=10; -- 更新数据

在没有提交事务时，数据库中的数据不会真正的发生变化



commit;-- 提交事务

事务提交后，数据库中的数据才会真正的变化



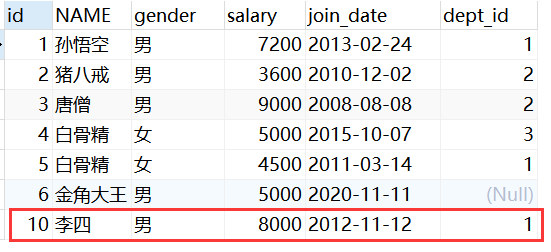
**示例2**

start transaction; --开启事务

update emp set name='王五' where id=10; -- 更新数据

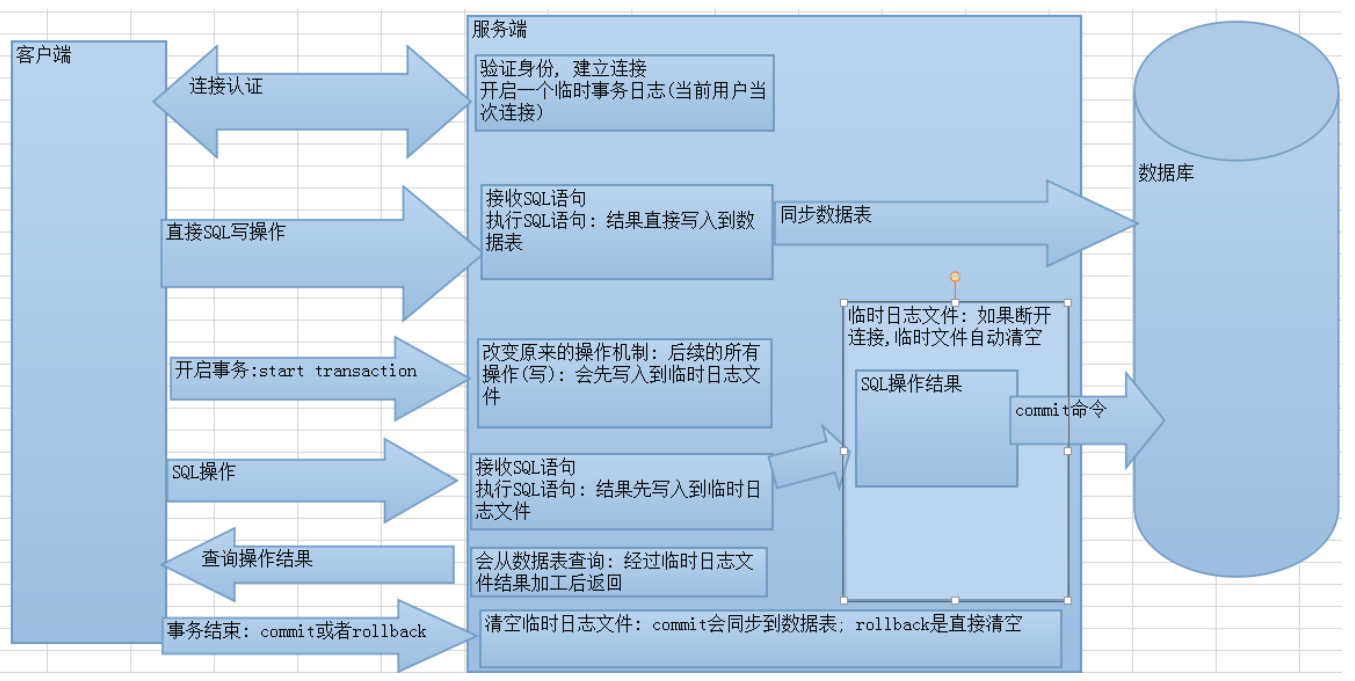
如果不提交事务，而是回滚事务，数据库中的数据将不会变化

rollback;



#### 事务的原理

事务开启之后, 所有的操作都会临时保存到事务日志, 事务日志只有在得到commit命令才会同步到数据表中，其他任何情况都会清空事务日志(rollback，断开连接)



### （五）事务的并发异常

在事务并发操作时（例如多个客户端来操作同一个数据库中的同一张表）。可能会出现如下的问题。

|  |  |
| --- | --- |
| **异常** | **描述** |
| 回滚丢失 | 此种更新丢失是因为回滚的原因，所以叫做回滚丢失。 |
| 覆盖丢失(脏写) | 此种更新是因为更新被其他事务覆盖了，所以叫做覆盖更新 |
| 脏读 | 一个事务竟然读取到了另一个事务尚未提交的数据 |
| 不可重复读 | 同一个事务对同一行数据执行了两次或以上的查询，结果却不同 |
| 幻读 | 幻读和不可重复读有点像，只是针对的一般是数据的数量 |

当然这些并发的异常可以说是由数据库的隔离级别引起的，或者说通过调整数据库的隔离级别可以防止以上并发异常的发生。

### （六）隔离级别

数据库的事务隔离级别主要包括：

**读未提交——Read Uncommitted**

**读已提交——Read Committed**

**可重复读——Repeatable Read**

**串行化——Serializable**

隔离级别越高，数据库的效率就越低。

MySQL数据库InnoDB存储引擎（默认）的默认隔离级别是**Repeatable Read**

Oracle数据库默认的隔离级别是**Read Committed**

各种隔离级别能解决哪些并发异常

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **隔离级别/并发异常** | **回滚丢失** | **脏读** | **不可重复读** | **幻读** | **覆盖丢失** |
| **读未提交** | **×** | **×** | **×** | **×** | **×** |
| **读已提交** | **√** | **√** | **×** | **×** | **×** |
| **可重复读** | **√** | **√** | **√** | **×** | **√** |
| **串行化** | **√** | **√** | **√** | **√** | **√** |

#### 1、隔离级别的查看和修改

##### （1）MySQL8以前版本

**查看**

select @@session.tx\_isolation

**修改**

set @@session.tx\_isolation=’隔离级别’

##### MySQL8

**查看**

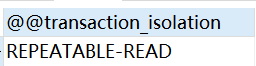
select @@transaction\_isolation

**修改**

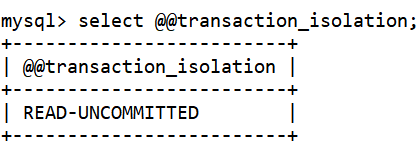
set session transaction isolation level 隔离级别

**示例**

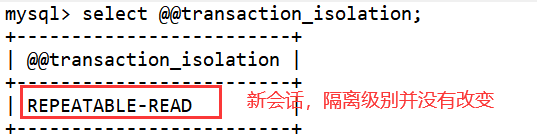
select @@transaction\_isolation



set session transaction isolation level read uncommitted;



当然，修改隔离级别，只是修改当前会话的隔离级别。如果打开另外一个会话查看，隔离级别没有改变。



#### 2、隔离级别应用

**准备数据**

create database tx character set utf8;

create table stu(

id int primary key auto\_increment,

name varchar(20),

age int

);

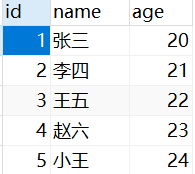
insert into stu values(null,'张三',20);

insert into stu values(null,'李四',21);

insert into stu values(null,'王五',22);

insert into stu values(null,'赵六',23);

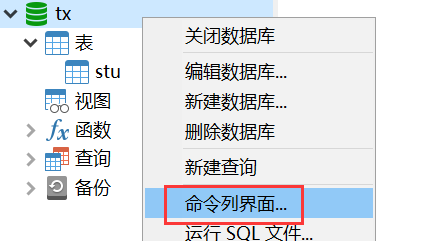
insert into stu values(null,'小王',24);

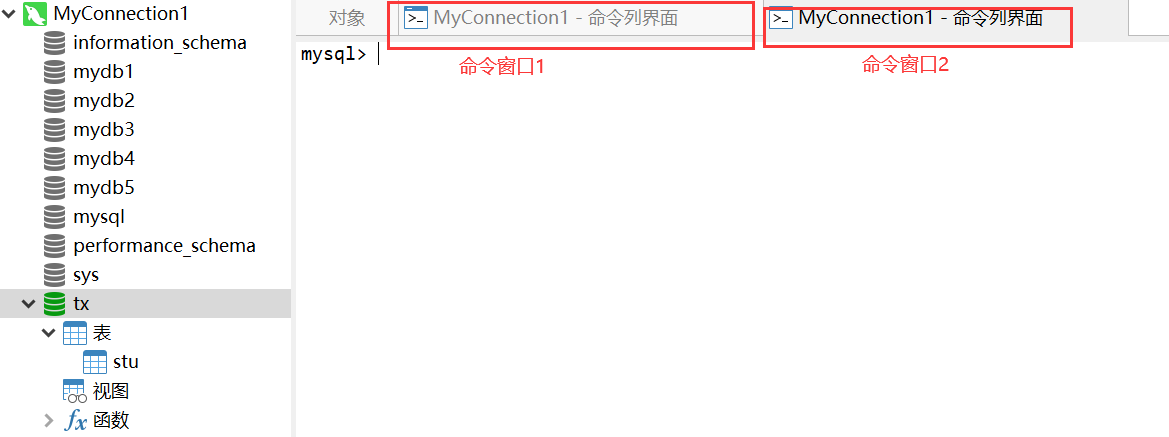


##### （1）Read Uncommitted

###### a、测试脏读

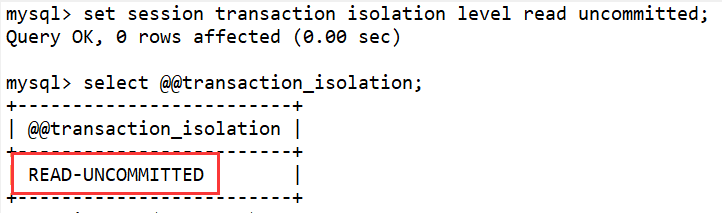
在Navicat中开启两个命令窗口，每一个窗口就相当于一个线程。





**1、把两个命令窗口的隔离级别都修改成 Read Uncommitted**

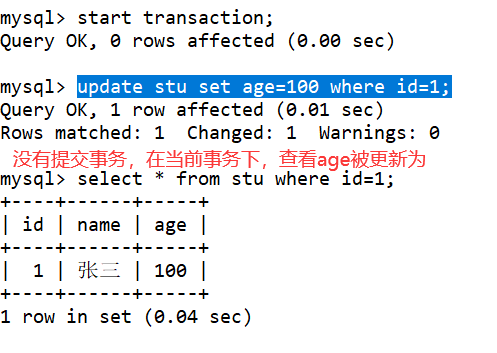
set session transaction isolation level read uncommitted



**2、在窗口1开启事务，更新id为1的年龄为100，不提交事务。**

start transaction;

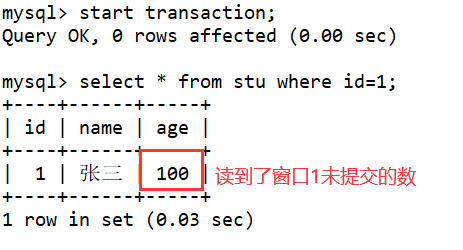
update stu set age=100 where id=1;



1. **在命令窗口2中，查看id为1的数据**

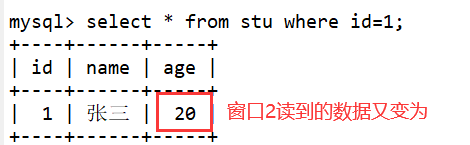
start transaction;

select \* from stu where id=1;



1. **如果窗口1回滚，窗口2读到的数据又会发生变化**



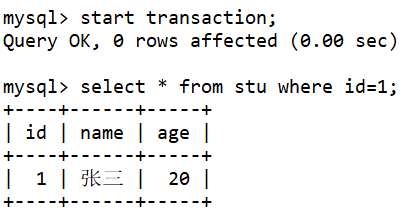


###### b、测试不可重复读

**窗口1开启事务查询id为1的数据**

start transaction;

select \* from stu where id=1;

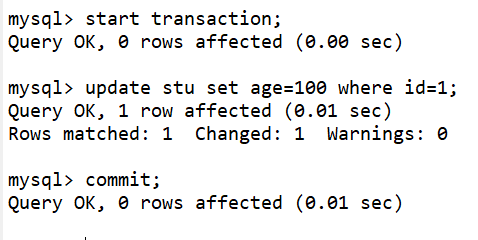


**2、窗口2开启事务，更新id为1的age为100**

start transaction;

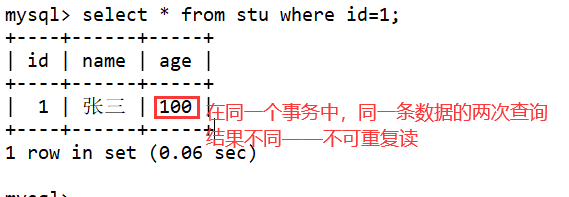
update stu set age=100 where id=1;

commit;



1. **回到窗口1，还是在原来的事务中，查询id为1的数据**

select \* from stu where id=1;

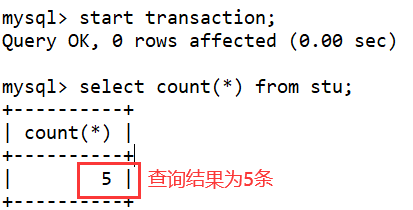


###### c、测试幻读

**1、在窗口1中开启事务，查询stu表的数据总条数**

start transaction;

select count(\*) from stu;

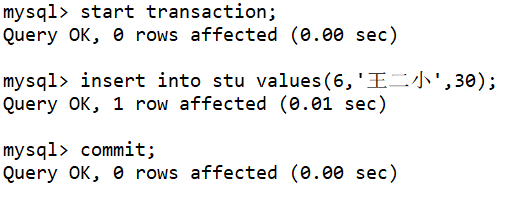


**2、在窗口2中开启事务，插入一条新数据**

start transaction;

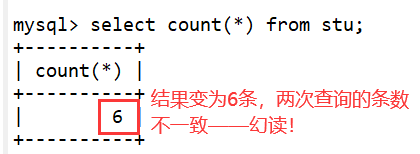
insert into stu values(6,'王二小',30);

commit;



**3、回到窗口1，还是在当前事务中，再次查询数据条数**

select count(\*) from stu;



###### d、测试丢失更新

关于丢失更新的说明

**回滚丢失**

|  |  |
| --- | --- |
| **事务A** | **事务B** |
| 查询数据得到初始值20 | 查询数据得到初始值20 |
| 更新为20+1=21 | 更新为20+2=22 |
| 提交 | 失败回滚 |
|  | 数据回到了初始状态20，导致事务A的更新丢失了，由于回滚导致的丢失叫做回滚丢失 |

**覆盖丢失**

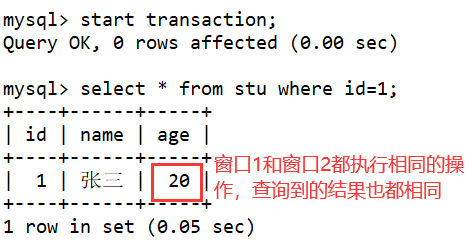
|  |  |
| --- | --- |
| **事务A** | **事务B** |
| 查询数据得到初始值20 | 查询数据得到初始值20 |
| 更新为20+1=21 | 更新为20+2=22 |
| 提交 | 提交 |
|  | 最终结果变为22，由于事务B是从20开始计算，并不知道数据已经发生了变化，所以导致事务A的更新被覆盖了，所以叫做覆盖更新 |

**测试回滚丢失**

1. **窗口1和窗口2都开启事务，查询id为1的数据的信息**

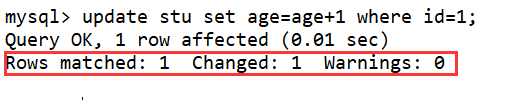
start transaction;

select \* from stu where id=1;



1. **窗口1更新id为1的age=age+1**

update stu set age=age+1 where id=1;



操作成功！！！

1. **窗口2更新id为1的age=age+2**

update stu set age=age+2 where id=1;



**会发现窗口2的更新会阻塞！这是因为在mysql数据库中，在窗口1更新这条数据没有提交时，对该条数据加锁了，导致窗口2无法更新这条数据。只有当窗口1条件是，窗口2的更新会自动执行。**

**窗口1**

**commit;**

**窗口2**

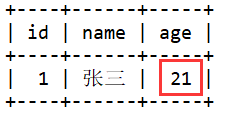


1. **窗口2回滚**

rollback;

1. **回到窗口1查询数据**

select \* from stu where id=1;



**我们会发现，并没有出现我们预期的丢失更新的情况！**

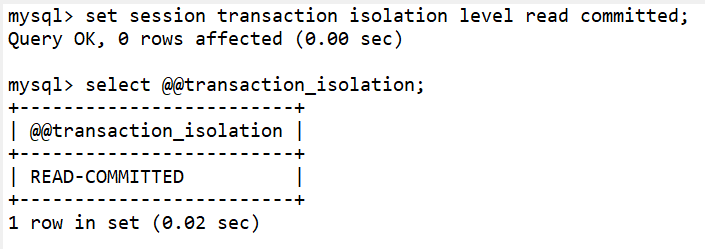
**这是因为MySQL的锁引起的，无论使用哪种隔离级别都无法演示出丢失更新的情况！除非关闭MySQL的锁。**

##### （2）Read Committed

###### a、测试脏读

**1、把两个命令窗口的隔离级别都修改成 Read Committed**

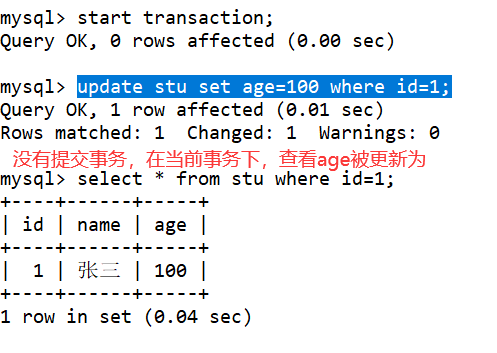
set session transaction isolation level read committed



**2、在窗口1开启事务，更新id为1的年龄为100，不提交事务。**

start transaction;

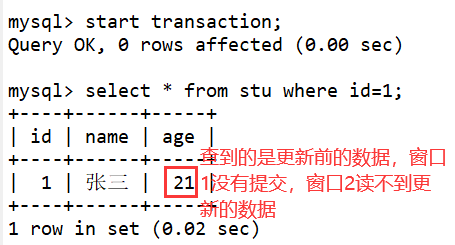
update stu set age=100 where id=1;



**3、在命令窗口2中，查看id为1的数据**

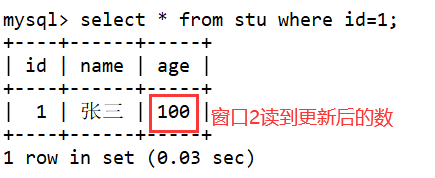
start transaction;

select \* from stu where id=1;



**4、窗口1提交，窗口2才能读到数据变化**



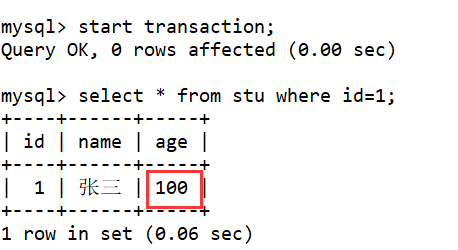


###### b、测试不可重复读

**窗口1开启事务查询id为1的数据**

start transaction;

select \* from stu where id=1;

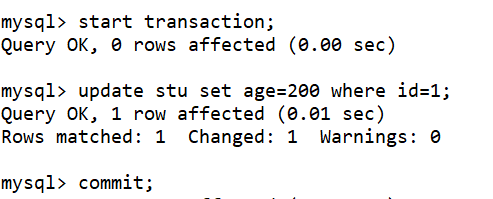


**2、窗口2开启事务，更新id为1的age为200**

start transaction;

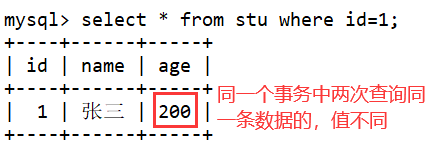
update stu set age=200 where id=1;

commit;



**3、回到窗口1，还是在原来的事务中，查询id为1的数据**

select \* from stu where id=1;



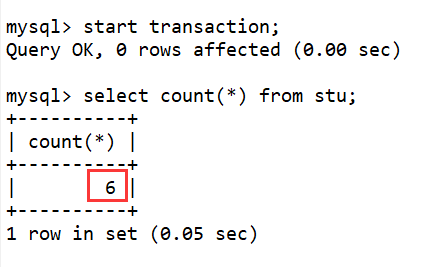
**不可重复读**

###### c、测试幻读

**1、在窗口1中开启事务，查询stu表的数据总条数**

start transaction;

select count(\*) from stu;

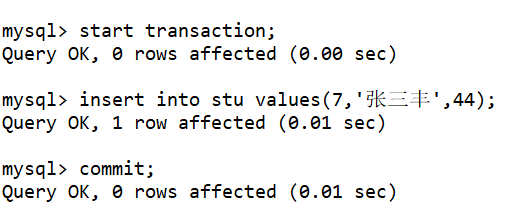


**2、在窗口2中开启事务，插入一条新数据**

start transaction;

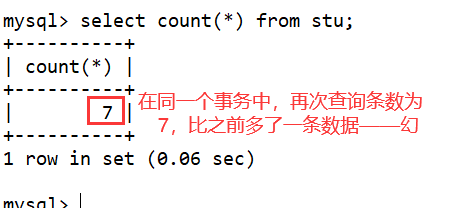
insert into stu values(7,'张三丰',44);

commit;



**3、回到窗口1，还是在当前事务中，再次查询数据条数**

select count(\*) from stu;

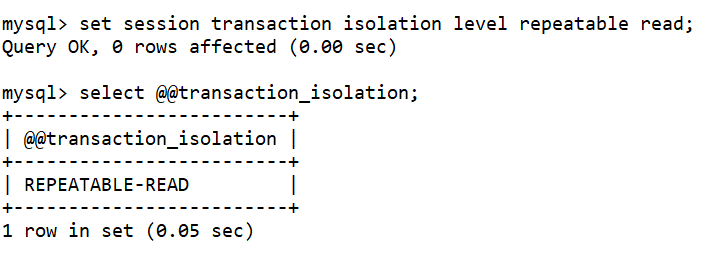


##### （3）Repeatable Read

###### a、测试脏读

**1、把两个命令窗口的隔离级别都修改成** Repeatable Read

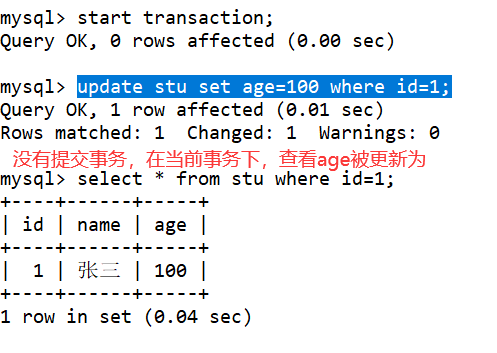
set session transaction isolation level repeatable read;



**2、在窗口1开启事务，更新id为1的年龄为100，不提交事务。**

start transaction;

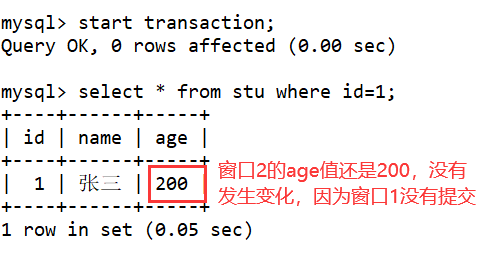
update stu set age=100 where id=1;



**3、在命令窗口2中，查看id为1的数据**

start transaction;

select \* from stu where id=1;



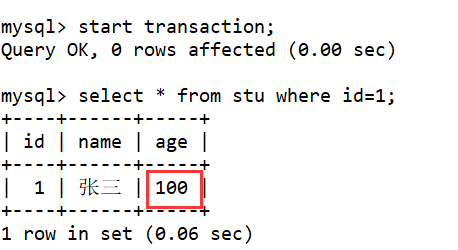
**不存在脏读**

###### b、测试不可重复读

**窗口1开启事务查询id为1的数据**

start transaction;

select \* from stu where id=1;

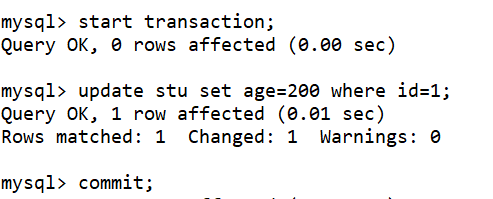


**2、窗口2开启事务，更新id为1的age为200**

start transaction;

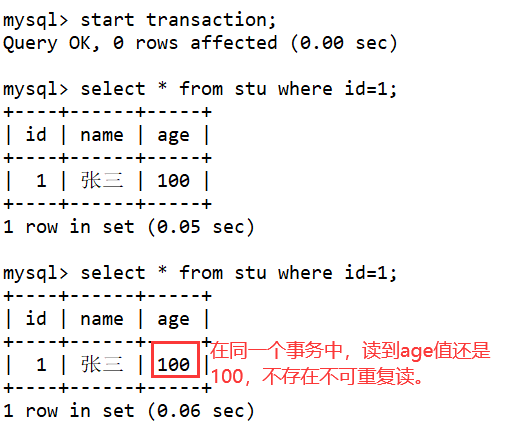
update stu set age=200 where id=1;

commit;



**3、回到窗口1，还是在原来的事务中，查询id为1的数据**

select \* from stu where id=1;

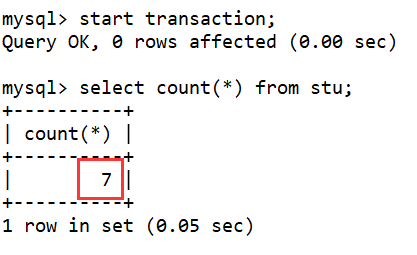


###### c、测试幻读

**1、在窗口1中开启事务，查询stu表的数据总条数**

start transaction;

select count(\*) from stu;

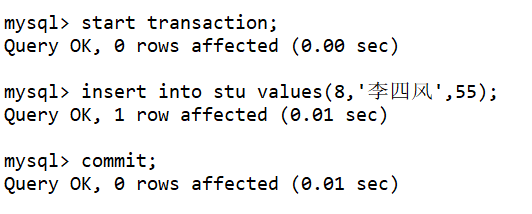


**2、在窗口2中开启事务，插入一条新数据**

start transaction;

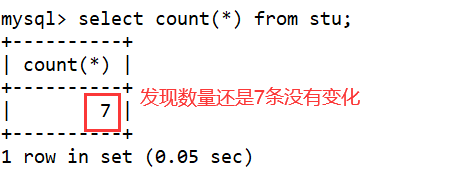
insert into stu values(8,'李四风',55);

commit;



**3、回到窗口1，还是在当前事务中，再次查询数据条数**

select count(\*) from stu;

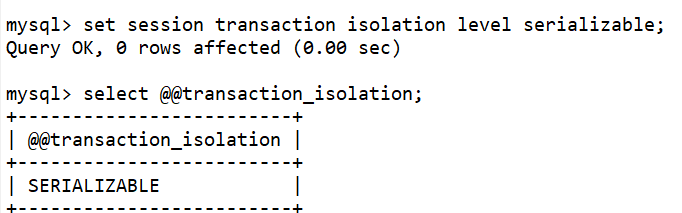


**我们发现，MySQL在Repeatable Read的隔离级别下，也不存在幻读了。这是因为MySQL中采用了next-key和gap-lock的机制的问题，这里不进行深入的探讨**

##### （4）Serializable

**1、窗口1和窗口2的隔离级别都设置为Serializable**

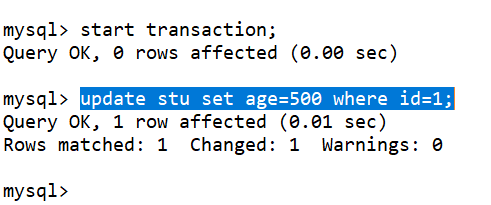
set session transaction isolation level serializable;



**2、窗口1开启事务，更新id为1的age为500，不提交事务**

start transaction;

update stu set age=500 where id=1;



**3、窗口2中开启事务查询id为1的数据**

select \* from stu where id=1；



**我们会发现，当窗口1不提或回滚结束事务时，窗口2不能进行任何操作。因此就也不存在脏读、不可重复读和幻读。**

## 十三、DCL

我们现在默认使用的都是root用户，超级管理员，拥有全部的权限。但是，一个公司里面的数据库服务器上面可能同时运行着很多个项目的数据库。所以，我们应该可以根据不同的项目建立不同的用户，分配不同的权限来管理和维护数据库。用户和权限的管理可以通过DCL（Data Control Language）来实现。

### 创建用户

**语法**

CREATE USER '用户名'@ '主机名' IDENTIFIED BY '密码'

1、`用户名`：将创建的用户名

2、`主机名`：指定该用户在哪个主机上可以登陆，如果是本地用户可用localhost，如果想让该用户可以从任意远程主机登陆，可以使用通配符%

3、`密码`：该用户的登陆密码，密码可以为空，如果为空则该用户可以不需要密码登陆服务器

**示例**

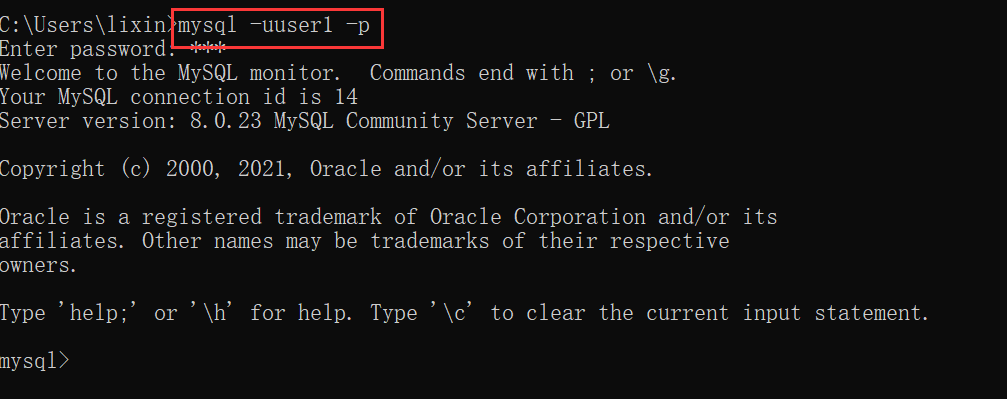
-- user1用户只能在localhost这个IP登录mysql服务器

CREATE USER 'user1'@'localhost' IDENTIFIED BY '123';

-- user2用户可以在任何电脑上登录mysql服务器

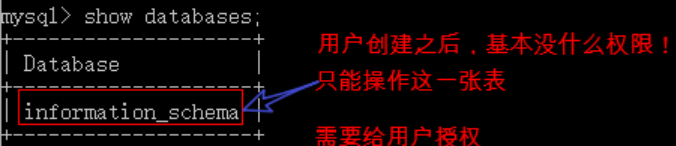
CREATE USER 'user2'@'%' IDENTIFIED BY '123';

使用新创建user1登录



### 授权用户

用户创建之后，基本没什么权限！需要给用户授权。



**语法**

GRANT 权限1, 权限2... ON 数据库名.表名 TO '用户名'@'主机名';

1、`GRANT` 授权关键字

2、授予用户的权限，如`SELECT`，`INSERT`，`UPDATE`等。如果要授予所的权限则使用`ALL`

3、 `数据库名.表名`：该用户可以操作哪个数据库的哪些表。如果要授予该用户对所有数据库和表的相应操作权限则可用表示，如`\*.\*`

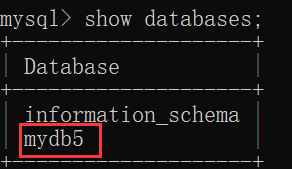
4、 `'用户名'@'主机名'`: 给哪个用户授权

**示例1**

给user1用户分配对mydb5这个数据库操作的权限。

GRANT CREATE,ALTER,DROP,INSERT,UPDATE,DELETE,SELECT ON mydb5.\* TO 'user1'@'localhost';

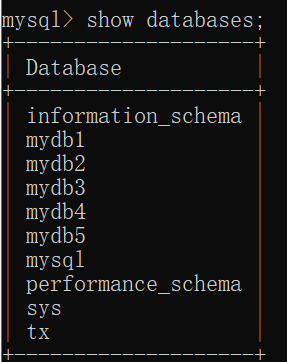
**注意要使用root用户进行授权**



**示例2**

给user2用户分配对所有数据库操作的权限。

GRANT ALL ON \*.\* TO 'user2'@'%';



### 查看权限

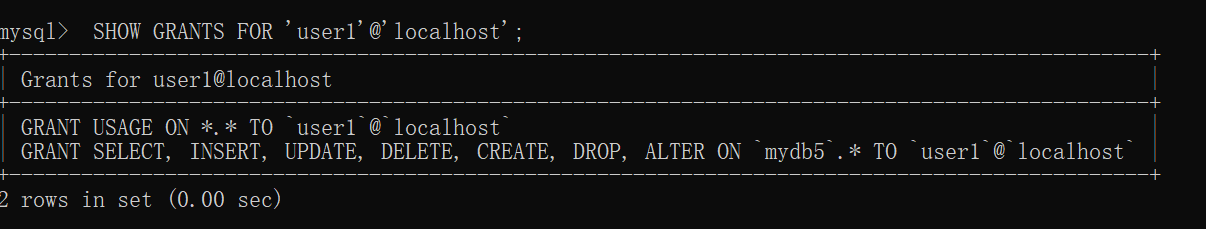
**语法**

SHOW GRANTS FOR '用户名'@'主机名';

**示例**

查看user1用户的权限

SHOW GRANTS FOR 'user1'@'localhost';



### 删除用户

**语法**

DROP USER '用户名'@'主机名';

**示例**

删除user2

DROP USER 'user2'@'localhost';

### 修改管理员密码

**语法**

mysqladmin -uroot -p password 新密码 -- 新密码不需要加上引号

**示例**

mysqladmin -uroot -p password 123456

输入老密码

**注意：要在未登录的情况下操作**

### 修改普通用户密码

**语法**

set password for '用户名'@'主机名' ='新密码';

**示例**

set password for 'user1'@'localhost' = '666666';

**注意：要在登录情况下操作**

## 索引

### 索引概述

MySQL官方对索引的定义为：索引（Index）是帮助MySQL高效获取数据的数据结构。

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。更通俗的说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度。

一般来说索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。

### 优缺点

#### 1、优点

类似大学图书馆建书目索引，提高数据检索的效率，降低数据库的IO成本。

通过索引列对数据进行排序，降低数据排序的成本，降低了CPU的消耗。

#### 2、缺点

实际上索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录，所以索引列也是要占用空间的。

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每次更新添加了索引列的字段，都会调整因为更新所带来的键值变化后的索引信息。

索引只是提高效率的一个因素，如果你的MySQL有大数据量的表，就需要花时间研究建立最优秀的索引，或优化查询语句。

### 分类

分为普通索引、唯一索引和组合索引（多列索引）。

#### 普通索引

##### 特点

普通索引使用没有限制，允许在所有数据类型上创建，可以有重复值和空值。

一个索引只包含单个列，一个表可以有多个单列索引。

##### 创建普通索引

###### 建表时创建

**语法**

INDEX 索引名 (列名)

**示例**

create table student(

id int primary key,

name varchar(20),

age int,

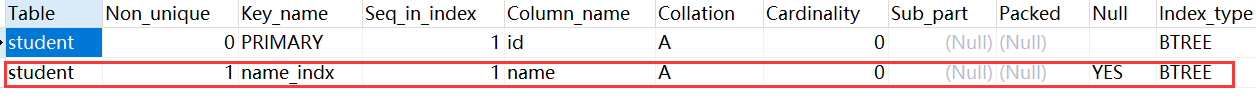
index name\_indx(name)

)



**测试是否创建成功**

show index from student;



###### 修改表方式创建

**语法**

ALTER TABLE 表名 ADD INDEX 索引名(列名);

**示例**

create table student2(

id int primary key,

name varchar(20),

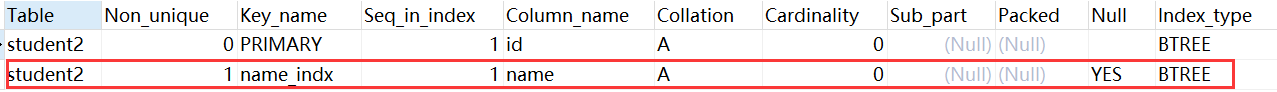
age int

)

ALTER TABLE student2 ADD INDEX name\_indx(name)

**测试是否创建成功**

show index from student2



###### 直接创建

**语法**

CREATE INDEX 索引名 ON 表名(列名);

**示例**

create table student3(

id int primary key,

name varchar(20),

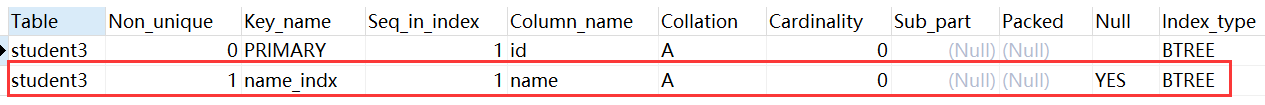
age int

)

CREATE INDEX name\_indx ON student3(name)

**测试是否创建成功**

show index from student3



#### 唯一索引

##### 特点

与普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须是唯一的，创建方法和普通索引类似。

##### 创建唯一索引

###### 建表时创建

**语法**

UNIQUE INDEX 索引名 (列名)

**示例**

create table student4(

id int primary key,

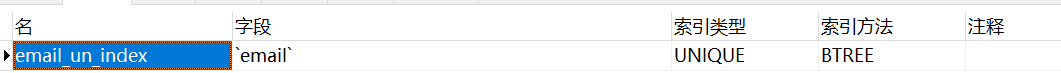
name varchar(20),

age int,

email varchar(200),

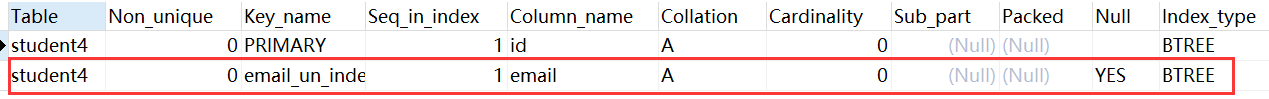
UNIQUE INDEX email\_un\_index(email)

)



**测试是否创建成功**

show index from student4



###### 修改表方式创建

**语法**

ALTER TABLE 表名 ADD UNIQUE INDEX 索引名(列名)

**示例**

create table student5(

id int primary key,

name varchar(20),

age int,

email varchar(200)

)

ALTER TABLE student5 ADD UNIQUE INDEX email\_un\_indx(email)



###### 直接创建

**语法**

CREATE UNIQUE INDEX 索引名 ON 表名(列名)

**示例**

create table student6(

id int primary key,

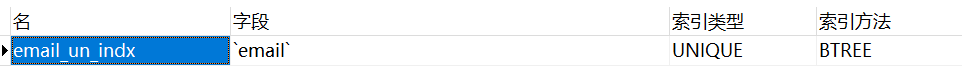
name varchar(20),

age int,

email varchar(200)

)

CREATE UNIQUE INDEX email\_un\_indx ON student6(email)



###### 添加唯一约束方式

**示例**

create table student7(

id int primary key,

name varchar(20),

age int,

email varchar(200) unique

)



添加了唯一约束的列，数据库会直接添加唯一索引。

#### 组合索引

##### （1）特点

同时在多个列上添加索引。当查询时，条件是添加了索引的列的组合，可以提高查询效率。

平时用的SQL查询语句一般都有比较多的限制条件，所以为了进一步榨取MySQL的效率，就要考虑建立组合索引。

##### （2）创建组合索引

###### 建表时创建（默认是普通组合索引）

**语法**

INDEX 索引名 (列名1,列名2...,列名n);

**示例**

create table student8(

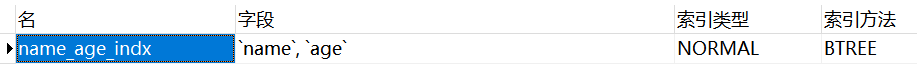
id int primary key,

name varchar(20),

age int,

**index name\_age\_indx(name,age)**

)



###### 修改表方式创建（默认是唯一组合索引）

**语法**

ALTER TABLE 表名 ADD UNIQUE INDEX 索引名 (列名1,列名2...,列名n)

**示例**

create table student9(

id int primary key,

name varchar(20),

age int

)

ALTER TABLE student9 ADD UNIQUE INDEX name\_age\_indx(name,age)



###### 直接创建（默认是普通组合索引）

**语法**

CREATE INDEX 索引名 ON 表名(列名1,列名2...,列名n)；

**示例**

create table student10(

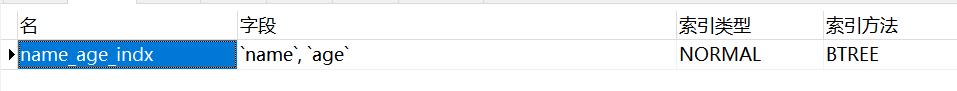
id int primary key,

name varchar(20),

age int

);

CREATE INDEX name\_age\_indx ON student10(name,age);



### 创建索引的条件

#### 1、创建索引

1、主键自动建立唯一索引。

2、频繁作为查询条件的字段应该创建索引。

3、查询中与其它表关联的字段，外键关系建立索引。

1. 查询中需要排序的字段，排序字段若通过索引去访问将大大提高排序速度。

5、普通/组合索引的选择问题，在高并发下倾向创建组合索引。

#### 2、不创建索引

1、频繁更新的字段不适合创建索引。

因为每次更新不单单是更新了记录还会更新索引，加重了I/O负担

2、Where条件里用不到的字段不创建索引

3、表记录太少。

4、经常增删改的表

5、如果某个数据列包含许多重复的内容，为它建立索引就没有太大的实际效果。

### （五）索引使用注意事项

#### 1、索引不会包含有NULL值的列

只要列中包含有NULL值都将不会被包含在索引中，复合索引中只要有一列含有NULL值，那么这一列对于此复合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为NULL。Is null和is not null 无法使用索引。

#### 2、使用短索引

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个CHAR(255)的列，如果在前10个或20个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作。

#### 3、索引列排序

MySQL查询只使用一个索引，因此如果where子句中已经使用了索引的话，那么order by中的列是不会使用索引的。因此数据库默认排序在符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

#### 4、like语句操作

一般情况下不鼓励使用like操作，如果非使用不可，如何使用也是一个问题。like “%aaa%” 不会使用索引而like “aaa%”可以使用索引。

#### 5、不要在列上进行运算

例如：select \* from users where YEAR(adddate)<2007，将在每个行上进行运算，这将导致索引失效而进行全表扫描，因此我们可以改成：select \* from users where adddate<’2007-01-01′。关于这一点可以围观：一个单引号引发的MySQL性能损失。

#### 6、MySQL使用不等于(!= 或者<>)时无法使用索引会导致全表扫描

在MySQL的查询语句的查询条件中如果使用了不等于，那么将索引失效，进行全表扫描方式查询。

#### 7、最左前缀法则

如果索引了多列，要遵守最左前缀法则。指的是查询从索引的最左前列开始并且不跳过索引中的列。

假设index(a,b,c)

|  |  |
| --- | --- |
| **where语句** | **索引是否被使用** |
| where a = 3 | Y,使用到a |
| where a = 3 and b = 5 | Y,使用到a，b |
| where a = 3 and b = 5 and c = 4 | Y,使用到a,b,c |
| where a = 3 and c = 5 | 使用到a， 但是c不可以，中间断了 |

## 存储过程

### （一）、什么是存储过程

存储过程是事先编译好并存储在数据库中的一段SQL语句的集合，可以重复直接调用。

### （二）、存储过程的优缺点

#### 1、存储过程优点

(1)存储过程在创建的时候直接编译，而sql语句每次使用都要编译，因此提高了执行效率。

(2)一条sql语句，可能需要访问几张表，对数据库连接好几次，存储过程只会连接一次。

(3)存储过程是安全的。数据库管理员可以向访问数据库中存储过程的应用程序授予适当的权限，而不向基础数据库表提供任何权限。

#### 2、存储过程缺点

(1)可移植性差。

(2)对于简单的sql语句，毫无意义。

(3)对于只有一类用户的系统安全性毫无意义。

(4)团队开发，标准不定好的话，后期维护很麻烦。

(5)对于开发和调试都很不方便。

(6)复杂的业务逻辑，用存储过程还是很吃力的。

### （三）创建存储过程

**语法**

delimiter 符号

create procedure 名称([参数])

begin

sql语句

end

符号

delimiter;

#### 1、简单的存储过程

**创建查询dept表中数据条数的存储过程**

Delimiter$$

create procedure dept\_count\_pro()

comment '查询dept表数据条数的存储过程'

begin

select count(\*)from dept;

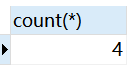
end;

$$

delimiter;

**调用存储过程：**

call dept\_count\_pro();



**说明**

comment '查询dept表数据条数的存储过程' 表示对存储过程的说明！

在MySQL中，默认使用;来表示一条语句的结束。在创建存储过程时，为了避免冲突，使用delimiter命令来改变MySQL数据库中结束符，将其改变为 $$。注意，使用delimiter命令可以自定义结束符，在创建完存储过程后使用delimiter;命令将结束符更改回MySQL原有的;。

#### 2、带输入参数的存储过程

向dept表中添加数据的存储过程

delimiter $$

create procedure dept\_insert\_pro(**in id int,in dept\_name varchar(20)**)

begin

declare v1 int default 0; /\* 定义变量\*/

declare v2 varchar(20); /\* 定义变量\*/

set v1=id; /\* 把参数赋给变量\*/

set v2 = dept\_name; /\* 把参数赋给变量\*/

insert into dept(id,name) values(v1,v2);

end

**$$**

delimiter;

call dept\_insert\_pro(5,'后勤部');



在声明和使用变量时，可以采用另外一种方式，可以省略declare

delimiter $$

create procedure dept\_insert\_pro2(**in id int,in dept\_name varchar(20)**)

begin

**set @v1=id;** /\* 声明变量并赋值\*/

**set @v2 = dept\_name;** /\* 声明变量并赋值\*/

insert into dept(id,name) values(**@v1,@v2**);

end

$$

delimiter;

call dept\_insert\_pro2(6,'安保部');



#### 3、带输出参数的存储过程

获取dept表中的id最大值

delimiter $$

create procedure dept\_getmaxid\_pro(**out maxid int**)

begin

select max(id) **into maxid** from dept;

end

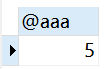
$$

delimiter;

call dept\_getmaxid\_pro(**@aaa**)

调用该存储过程，注意：输出参数必须是一个带@符号的变量

select **@aaa**



#### 4、带输入和输出参数的存储过程

delimiter $$

create procedure sp(in p1 int , out p2 int)

begin

**if** p1 = 1 **then**

set @v = 100;

**else**

set @v = 200;

**end if;**

/\* 语句体内可以执行多条sql，但必须以分号分隔 \*/

insert into dept(id) values(@v);

select max(id) into p2 from dept;

end

$$

delimiter;

call sp(1,@aaa)

select @aaa





**说明**

存储过程中，可以使用if then ... else ... end if; 结构来做条件判断语句。

#### 5、既做输入又做输出参数的存储过程

delimiter $$

create procedure sp4(inout p4 int)

begin

if p4 = 4 then

set @pg = 400;

else

set @pg = 500;

end if;

select @pg;

end

$$

delimiter;

set @pp = 4;

call sp4(@pp)



## 五、存储引擎

### （一）概述

存储引擎是MySQL中特有的术语，主要是指在MySQL中，用户可以根据应用的需求，选择如何存储、更新、查询和索引数据，是否使用事务等。

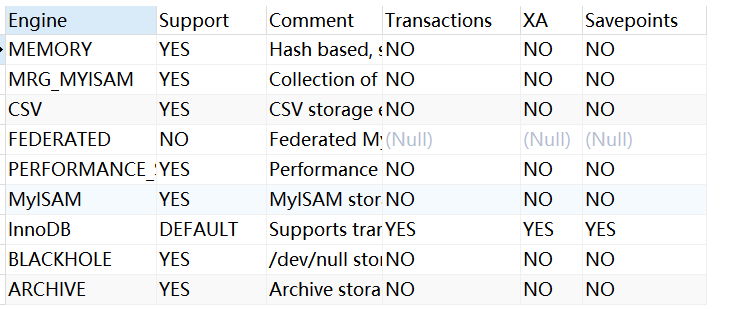
在MySQL中，支持多种不同类型的存储引擎，从而满足用户使用各种方式存储数据的需求。

### 类型

MySQL提供多种存储引擎，但不同版本的MySQL所提供的存储引擎类型稍有不同。可以使用SQL语句查看当前版本数据库中都提供了哪些存储引擎。

语句

show engines;



当前8.0.23包含9种存储引擎。

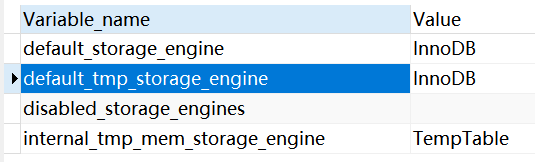
### 常见存储引擎简介

MySQL8.0.23中包括9种存储引擎。其中InnoDB、MyISAM、MRG\_MYISAM(又称为MERGE)、MEMORY、BLACKHOLE比较常见。

|  |  |
| --- | --- |
| **存储引擎** | **特点** |
| InnoDB | 支持事务处理，保证事务安全；支持外键；采用行锁 |
| MyISAM | 不支持事务的完整性和并发性，采用表锁；访问速度快；占用空间少 |
| MRG\_MYISAM | 将多个MyISAM表合成一个单表；不支持事务处理；采用表锁 |
| MEMORY | 使用存储在内存中的内容创建表，所有数据存放在内存中，因此读取速度快，但容易丢失数据 |
| BLACKHOLE | 不用于存储数据，而是将数据存入binlog中 |

查看当前系统使用的存储引擎

show variables like '%storage\_engine%';



### （四）InnoDB存储引擎介绍

在MySQL被Oracle收购之后，其开发重心转移到InnoDB存储引擎上，并且在MySQL5.5之后默认存储引擎改为了InnoDB。

#### 1、存储结构

在InnoDB存储引擎中，所有数据都被逻辑地存储在表空间（tablespace）中，表空间又由段（segment）、区（extent）、页（page）组成。其中，一个表空间包括多个段，一个段由多个区组成，一个区内又可以分为多个页。

##### 表空间

表空间可以分为系统表空间、独立表空间和临时表空间。

当数据库初始化时，一些初始信息都是被存在一个叫做ibdata1的表空间文件中。对于系统表空间，其大小随着数据量的增加会自动扩展，一般默认自动扩展大小是64M。

独立表空间意味着数据库中的每个表都有自己的表空间文件，使用独立表空间只需将innodb\_file\_per\_table参数设置为1即可。

临时表空间是从5.7版本开始从系统表空间中抽离出来而形成的，将临时表中的相关信息保存在其中。

##### 段

在InnoDB存储引擎中，常见的段主要有数据段、索引段、回滚段。

##### 区

区由连续的页构成，在物理存储上由系统分配一段连续的空间进行存储。每个区由64个连续的页构成，大小都是1MB。

##### 页

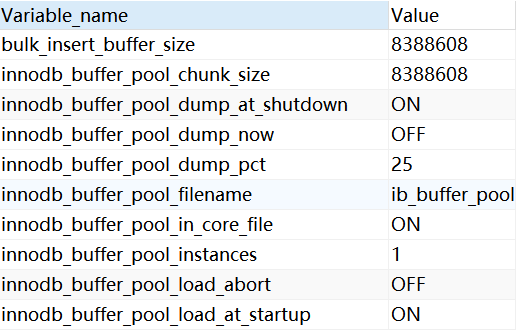
页是InnoDB存储引擎中最小的存储分配单位，默认大小为16KB。

#### 2、InnoDB内存结构

在MySQL中，当使用InnoDB存储引擎时，其内存结构与Oracle数据库的内存结构类似，分为系统全局区（SGA）和程序缓冲区（PGA）两部分。

用户可以查看InnoDB存储引擎的内存结构参数，SQL语句如下：

show variables like '%buffer%';



**系统全局区常用参数**

|  |  |
| --- | --- |
| **参数** | **作用** |
| innodb\_buffer\_pool | 缓存表中的数据、索引、插入缓冲、数据字典等信息 |
| innodb\_log\_buffer\_size | 设置事务在内存中的缓存大小 |
| innodb\_sort\_buffer\_size | 设置InnoDB进行排序时的缓存大小 |

**程序缓存区常用参数**

|  |  |
| --- | --- |
| **参数** | **作用** |
| sort\_buffer\_size | 设置用于SQL语句在内存中排序的缓存大小 |
| join\_buffer\_size | 设置用于表连接的缓存大小 |
| read\_buffer\_size | 设置用于表顺序扫描的缓存大小 |
| read\_rnd\_buffer\_size | 设置用于随机读的缓存大小 |

#### InnoDB特性

InnoDB存储引擎有三个主要特性分别是插入缓冲（change buffer）、两次写（double write）、自适应哈希索引（adaptive hash index）。

##### 插入缓冲

对于插入缓存，首先根据插入的普通索引页在缓冲池里查找，如果找到则直接将待插入页插入表中，如果没找到则先将待插入页放入缓冲池中，与之前的缓冲池进行合并操作，当有多个插入进行时，可以将其合并到一个操作中，从而提高插入的性能。

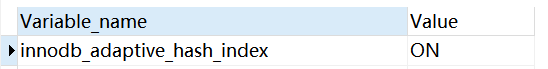
##### 两次写

当InnoDB缓冲池刷出新的“脏页”要写入磁盘文件时，由于两次写机制的存在，会首先将“脏页”写入两次写缓存中，随后两次写缓存以每次1MB的大小将数据写入两次写所在的磁盘表空间，并同时写入数据文件中。正是由于两次写的存在，保证了MySQL数据库中数据的安全性。

##### 自适应哈希索引

当使用InnoDB存储引擎时，会对索引的搜索进行监控，如果某个查询能使用哈希索引来优化查询索引速度，那么数据库系统将会自动完成这项操作，而不需要人为操作来完成。用户可以查看是否打开了该功能，SQL语句如下：

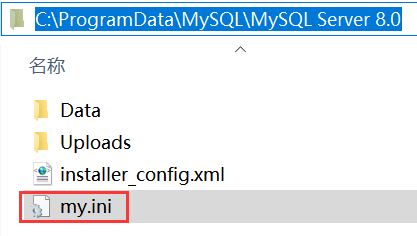
show variables like '%innodb\_adaptive\_hash\_index%';



### （五）修改MySQL的存储引擎

#### 修改my.ini文件

找到my.ini文件



通过文本编辑工具打开，可以进行修改



#### 建表时指定表的存储引擎

CREATE TABLE `emp` (

`id` int NOT NULL AUTO\_INCREMENT,

`NAME` varchar(10) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL,

`gender` char(1) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL,

`salary` double NULL DEFAULT NULL,

`join\_date` date NULL DEFAULT NULL,

`dept\_id` int NULL DEFAULT NULL,

PRIMARY KEY (`id`) USING BTREE

) **ENGINE = InnoDB** AUTO\_INCREMENT = 10 CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci ROW\_FORMAT = Dynamic;

## 五、综合练习

（1）

https://www.nowcoder.com/practice/53235096538a456b9220fce120c062b3?tpId=199&tags=&title=&difficulty=0&judgeStatus=0&rp=0&sourceUrl=%2Fexam%2Foj%3Fpage%3D1%26tab%3DSQL%25E7%25AF%2587%26topicId%3D199

select

  u.device\_id,

  u.university,

  count(q.question\_id) as question\_cnt,

  sum(if(q.result = 'right', 1, 0)) as right\_question\_cnt

from

  user\_profile u

  LEFT JOIN question\_practice\_detail q on q.device\_id = u.device\_id

  and month(q.date) = 8

where

  university = '复旦大学'

group by

  u.device\_id;

（2）

https://www.nowcoder.com/practice/d8a4f7b1ded04948b5435a45f03ead8c?tpId=199&tags=&title=&difficulty=0&judgeStatus=0&rp=0&sourceUrl=%2Fexam%2Foj%3Fpage%3D1%26tab%3DSQL%25E7%25AF%2587%26topicId%3D199

select

  difficult\_level,

  sum(if(q1.result = 'right', 1, 0)) / count(q1.question\_id) as correct\_rate

from

  user\_profile u

  inner join question\_practice\_detail q1 on u.device\_id = q1.device\_id

  inner join question\_detail q2 on q1.question\_id = q2.question\_id

where

  u.university = '浙江大学'

group by

  q2.difficult\_level

order by

  correct\_rate

（3）