# 第二讲 MySQL数据库

|  |  |  |  |
| --- | --- | --- | --- |
| **课程大纲** | **课程内容** | **学习效果** | **掌握目标** |
| MySQL | DQL | 掌握 | 掌握查询语句排序、单行函数、聚合函数、分组、limit查询等，并熟练使用 |
| 约束和策略 | 掌握 | 掌握主键、非空、唯一、缺省、外键约束的创建和使用方式，掌握主键自增策略的使用和截断表 |
| 表关系 | 掌握 | 掌握一对多、多对多、一对一的表关系。掌握各种关系下建表的规则 |
| 多表查询 | 掌握 | 掌握笛卡尔积、内连接、外连接和子查询 |
| 数据库备份与还原 | 掌握 | 掌握语句方式和可视化方式备份数据库和还原数据库 |

## 一、DQL数据查询语言

### （一）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

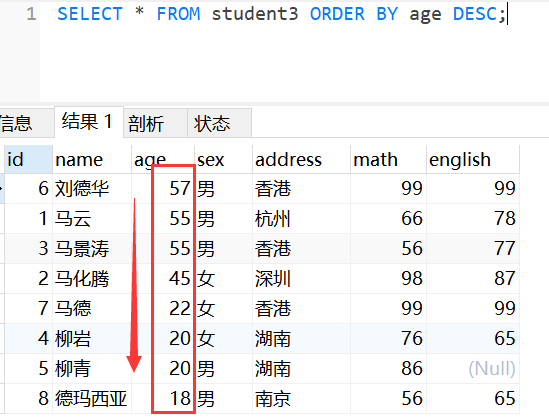
#### 单列排序

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。

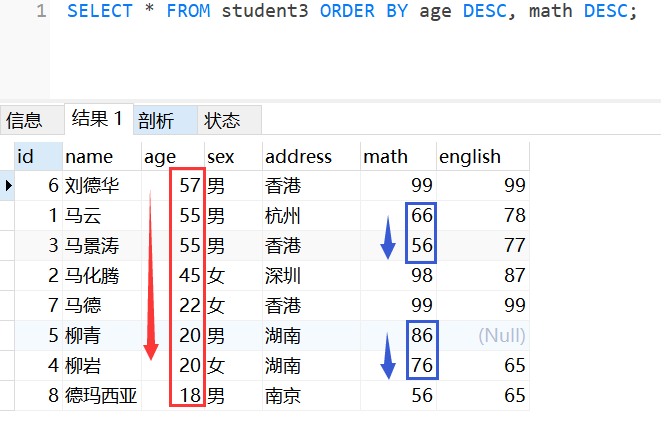
**语法**

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;



### 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 ConcatenatedString;

返回 SQLRunoobGooogleFacebook

##### 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

##### substr(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

##### 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)

在参数日期上增加相应的时间

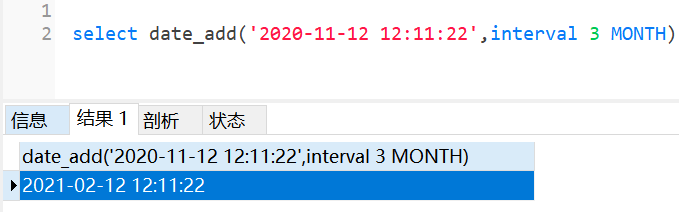
date 参数是合法的日期表达式。expr 参数是您希望添加的时间间隔。

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 |

**示例**

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



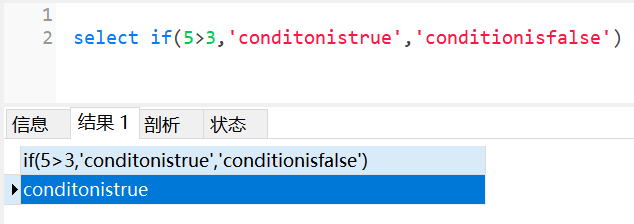
#### 流程控制函数

##### 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')

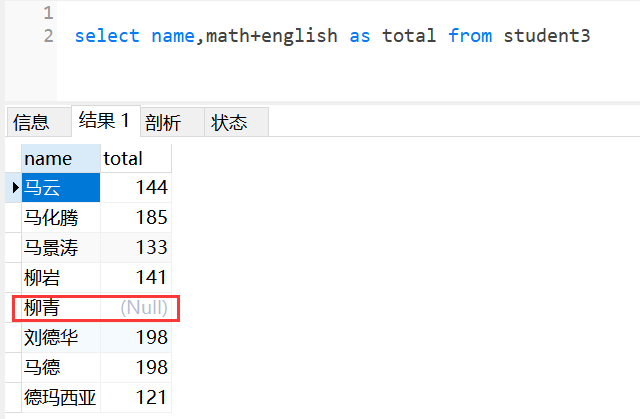


##### ifnull(expr1,expr2)

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

**示例**

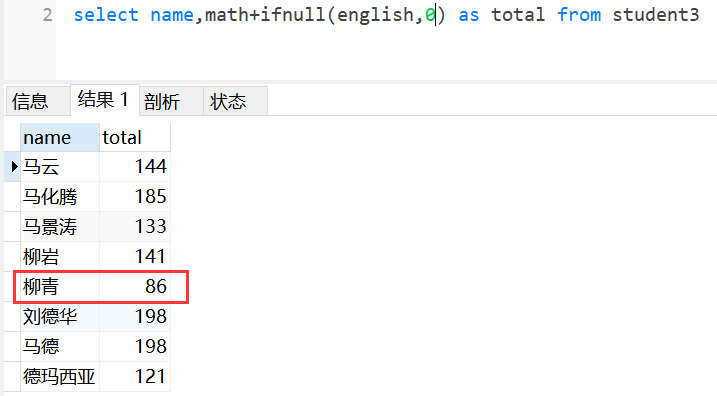
用此函数解决前面null运算结果为null的问题



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



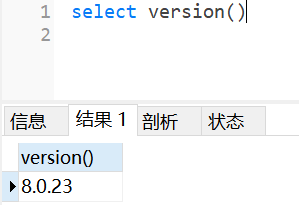
#### 其他函数

##### 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()

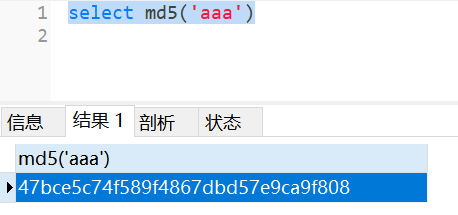


##### md5(expr)

返回参数md5加密后的值

**示例**

select md5('aaa')



### 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

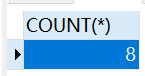
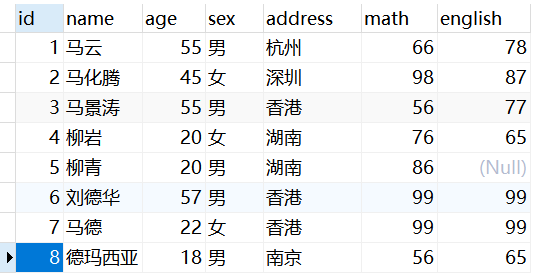
#### count函数

统计指定列记录数

**示例**

查询总条数

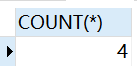
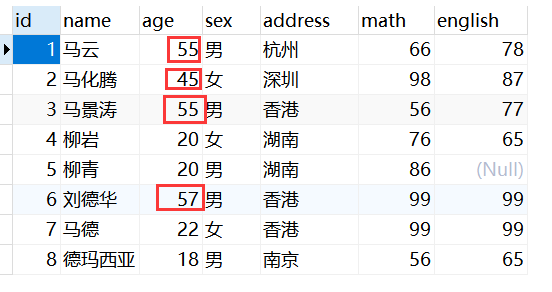
SELECT COUNT(\*) FROM student3;



**示例2**

查询年龄大于40的总人数

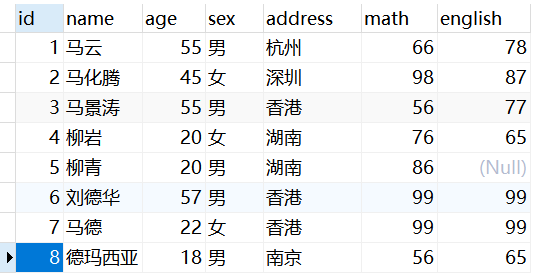
SELECT COUNT(\*) FROM student3 WHERE age>40;



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**

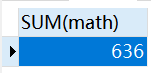
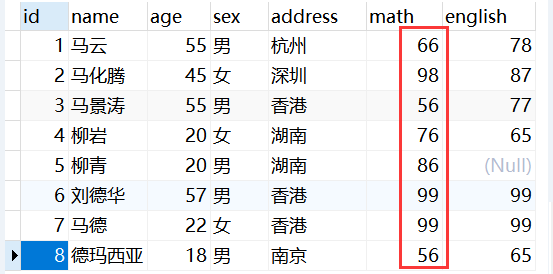
#### sum函数

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;



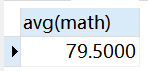
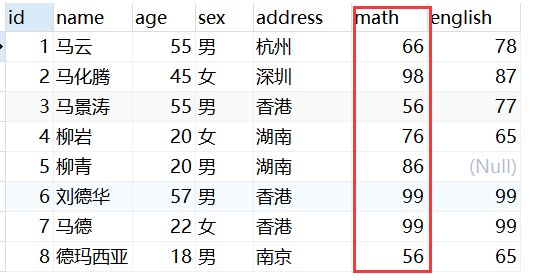
#### avg函数

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;



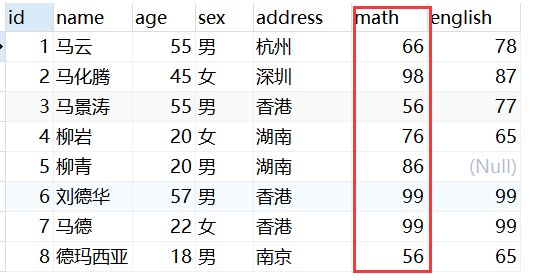
#### 4、max函数

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;



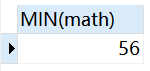
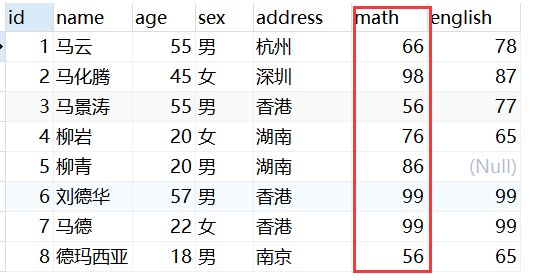
#### 5、min函数

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;



### 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];

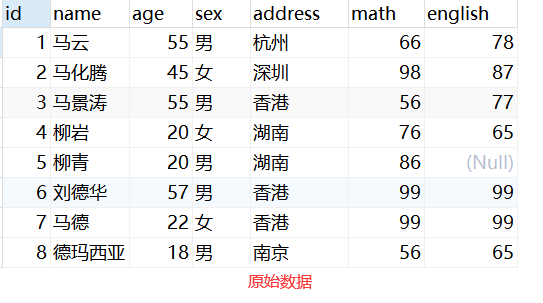
#### 1、分组查询

GROUP BY怎么分组的？**将分组字段结果中相同内容作为一组**

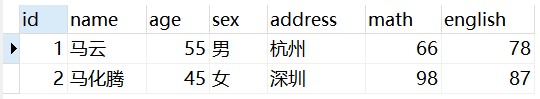
**示例1**

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**

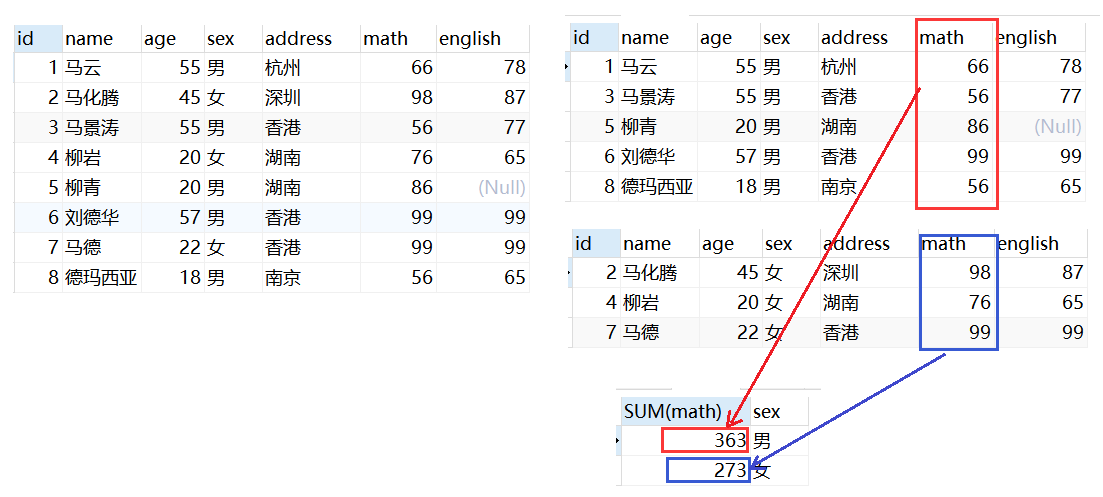
以上的操作是将分组字段结果中相同内容作为一组，并且返回每组的第一条数据。这样的操作显然没有什么意义。而且在有些数据库中（例如oracle）中，这样的语句是错误的！

**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。



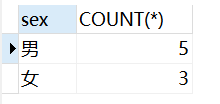
聚合函数是在分组后运行，统计每组的数据

**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;

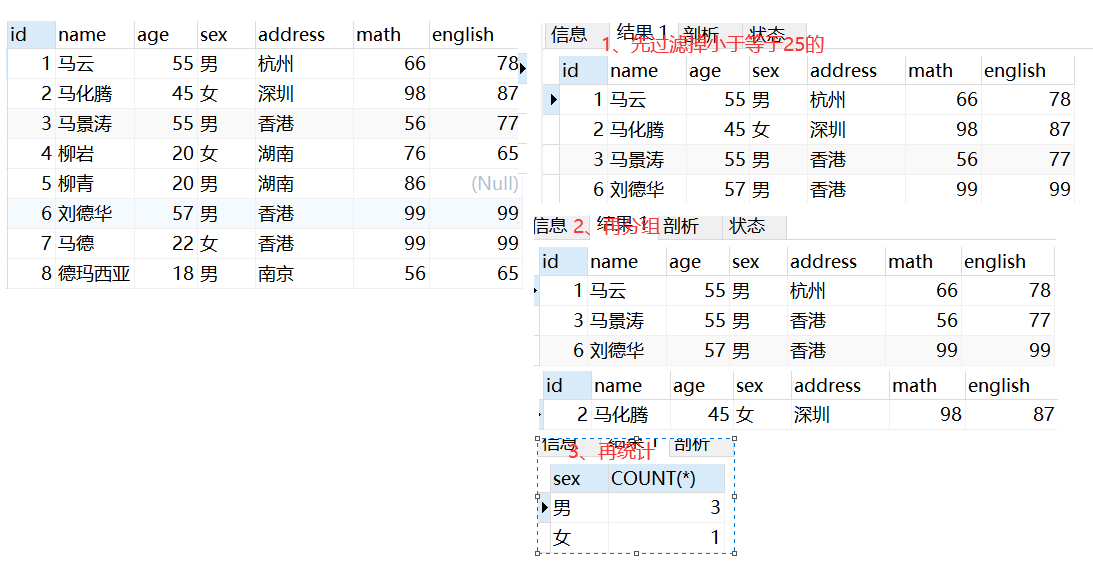


#### 2、分组时的条件过滤

**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

### （五）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;

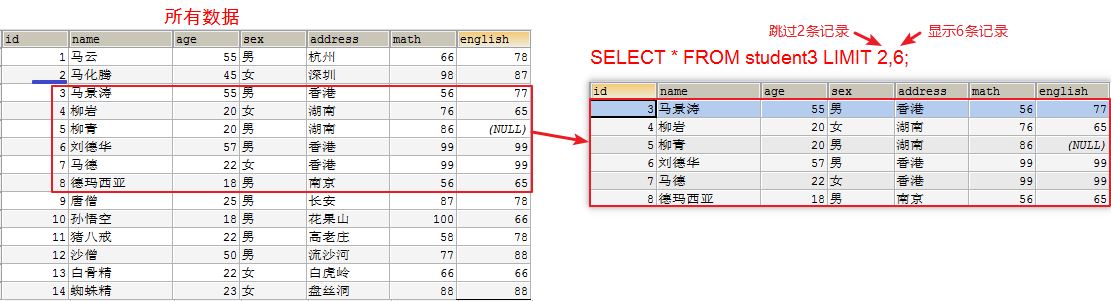
offset是指偏移量，可以认为是跳过的记录数量，默认为0，表示从哪开始

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。

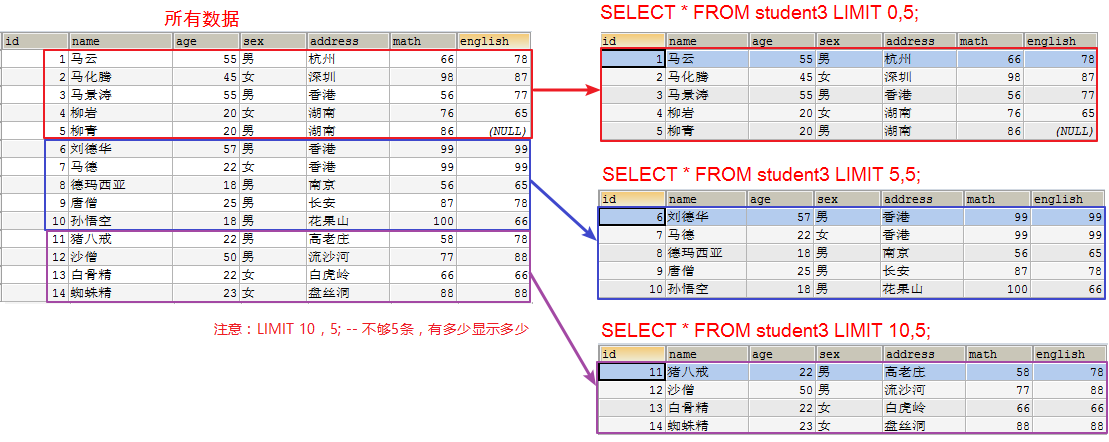
SELECT \* FROM student3 LIMIT 2,6;



**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。

### （六）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

## 约束和策略

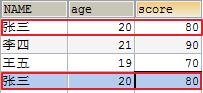
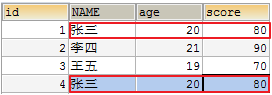
约束的作用：对表中的数据进行进一步的限制，保证数据的**正确性**、**有效性**和**完整性**。约束都是加在列上的，对一列的数据进行限制。

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 stu1(

id int **primary key**,

name varchar(20)

)

###### 方式二

**示例**

create table stu1(

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),

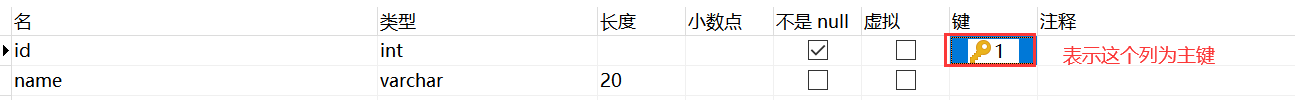
)

**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stu1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**

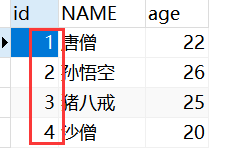
**示例**

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

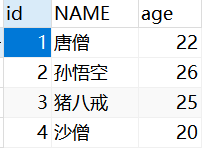
truncate表示截断表，会删除表中的全部数据，相当于删除表后重新创建该表。

**语法**

truncate table 表名

**示例**

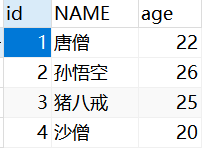
truncate table stu2



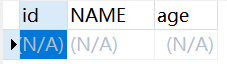
**DELETE和TRUNCATE对于主键自增策略的区别**

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

**示例**



delete from stu2;



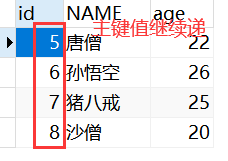
重新添加数据

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

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

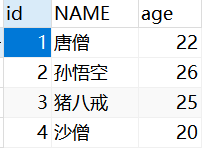
INSERT INTO stu2(NAME, age) VALUES ('猪八戒', 25);

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

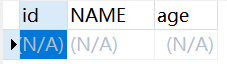


1. TRUNCATE 摧毁表，重建表，AUTO\_INCREMENT重置为1

**示例**



truncate table stu2;



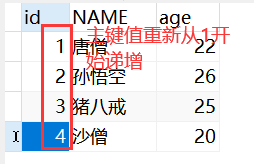
重新添加数据

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,

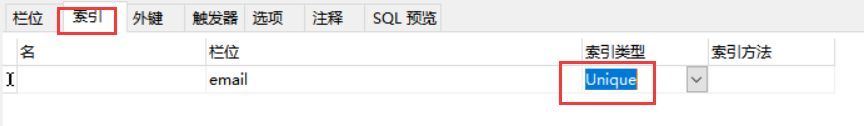
name varchar(20),

email varchar(200)

)

**alter table stu4 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 stu4 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,

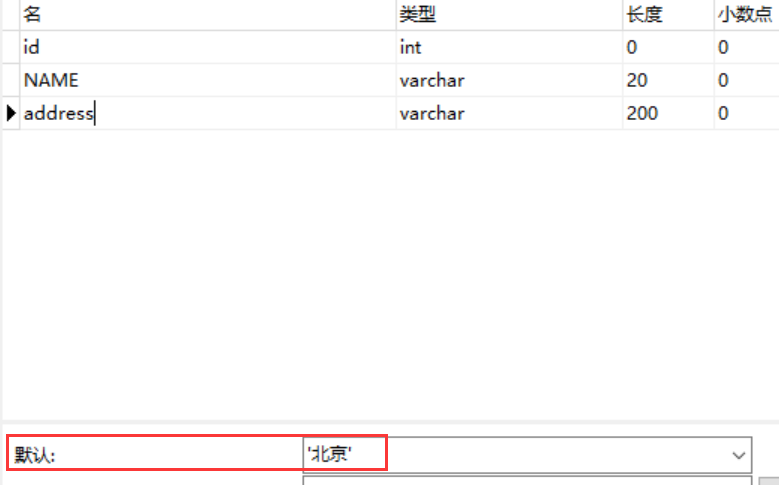
NAME VARCHAR(20),

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');

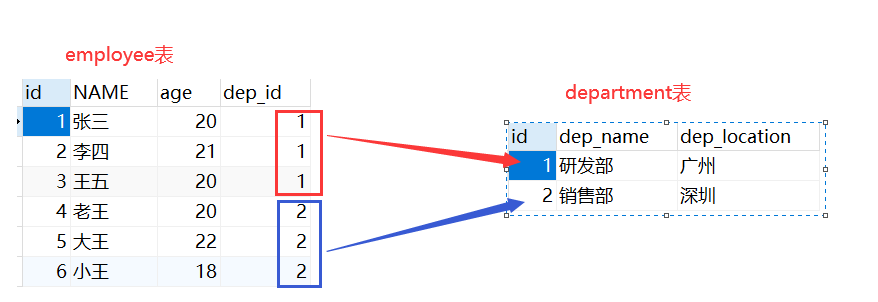
**缺点:表中出现了很多重复的数据(数据冗余)，如果要修改研发部的地址需要修改3个地方。**



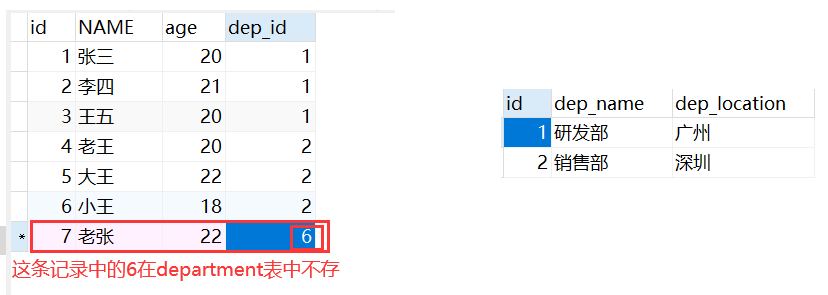
#### 2、解决方案

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

##### （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

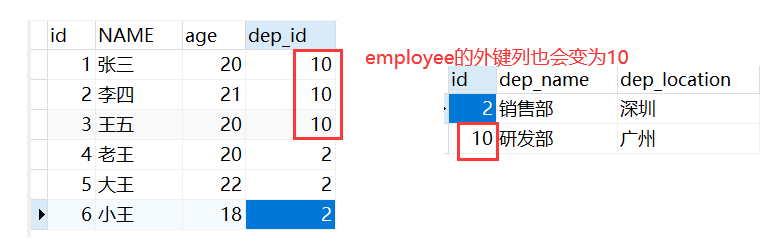
CREATE TABLE employee (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(30),  
 age INT,  
 dep\_id INT,  
 -- 添加外键约束,并且添加级联更新和级联删除  
 CONSTRAINT employee\_dep\_fk FOREIGN KEY (dep\_id) REFERENCES department(id) **ON UPDATE CASCADE ON DELETE CASCADE**);

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);

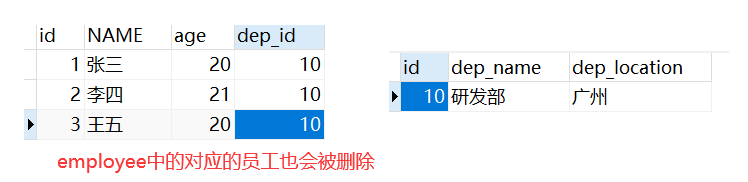
**把部门表中id等于1的部门改成id等于10**

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**

## 三、表关系

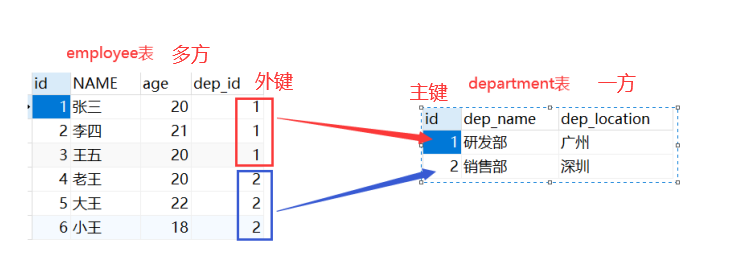
### （一）概念和分类

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

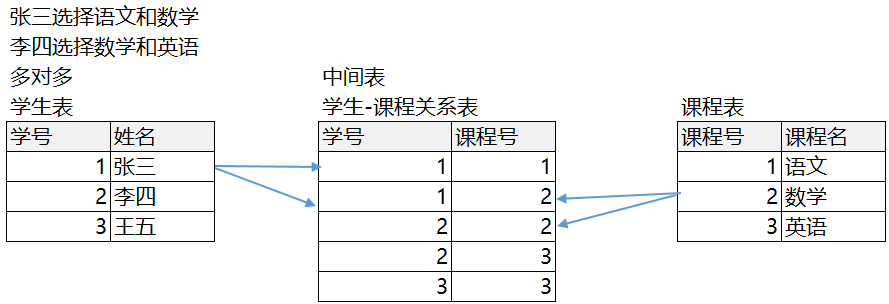
一对多建表原则: 在从表(多方)创建一个字段，字段作为外键指向主表(一方)的主键 

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

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**

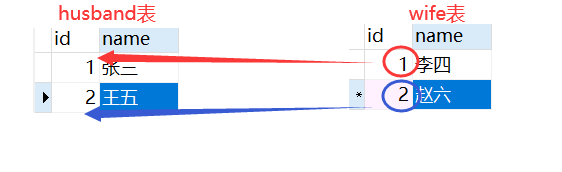
create table wife(

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)

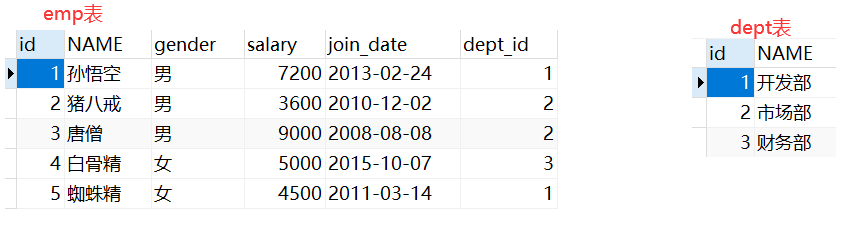
)



## 四、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。

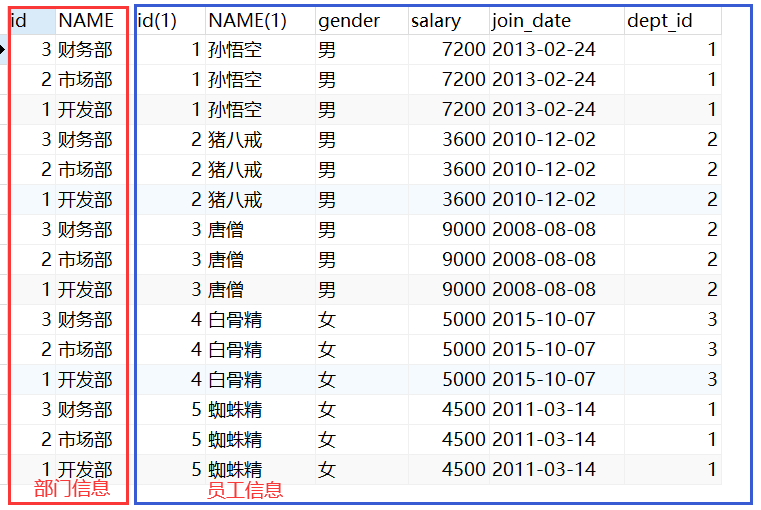
**语法**

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

**示例**

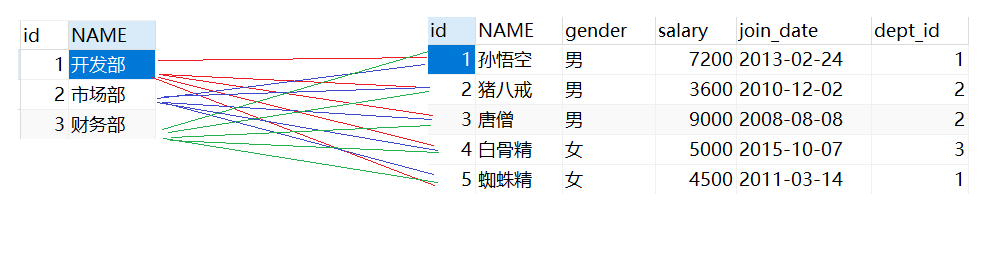
需求：查询部门和员工信息

SELECT \* FROM dept, emp;



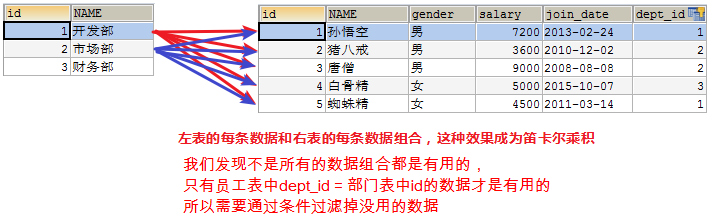
以上数据其实是左表的每条数据和右表的每条数据组合。左表有3条，右表有5条，最终组合后3\*5=15条数据。

**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。**

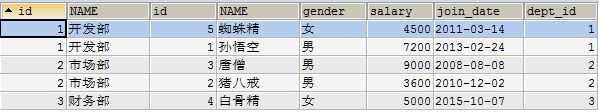


#### 2、如何清除笛卡尔积现象的影响

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。

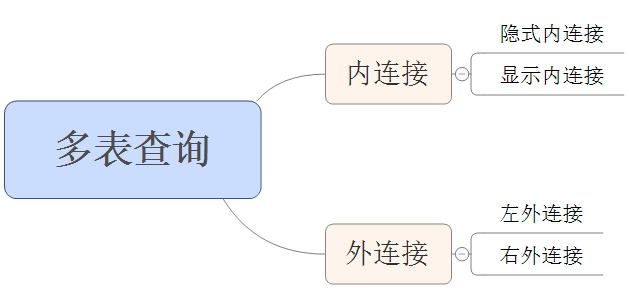


SELECT \* FROM dept, emp WHERE emp.`dept\_id`=dept.`id`;



### （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

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

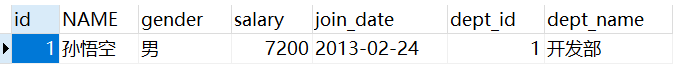
select e.name emp\_name,d.name dept\_name from emp e,dept d where e.dept\_id = d.id



**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'

#### 2、外连接

##### （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。

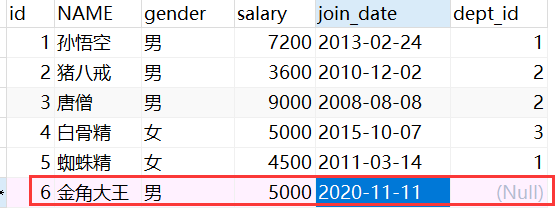
**语法**

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。

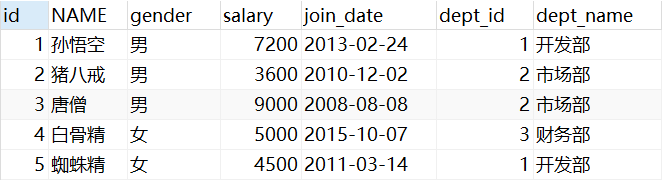
**示例**

新入职了一名员工‘金角大王’，但该员工没有分配部门。

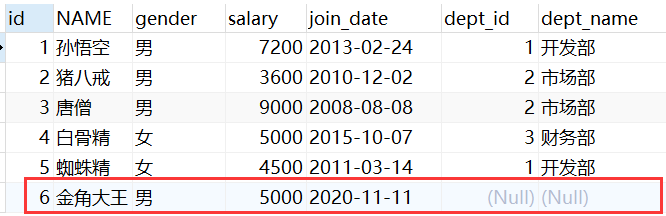


现在要查询员工的信息和所在部门的名称。如果使用内连接查询，没有部门的员工将不会显示，因为不满足连接条件。

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**

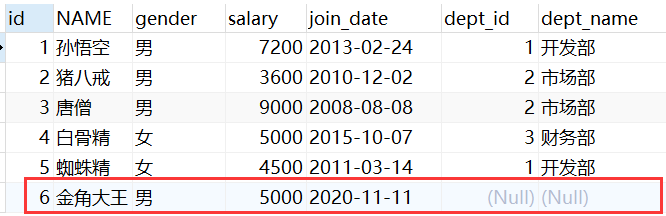
SELECT 字段名 FROM 左表 RIGHT OUTER JOIN 右表 ON 条件；

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。

### （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)

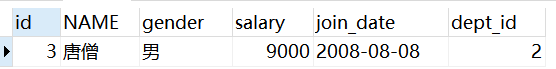
**示例**

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。

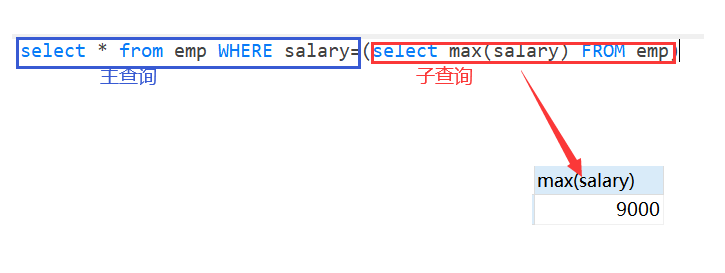
2、将子查询放在比较条件的右边。（非强制要求）

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**

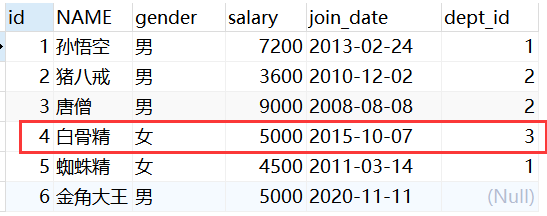
= > >= < <= != 等我们学习过的众多运算符都是单行运算符

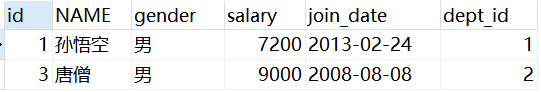
**示例**

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**

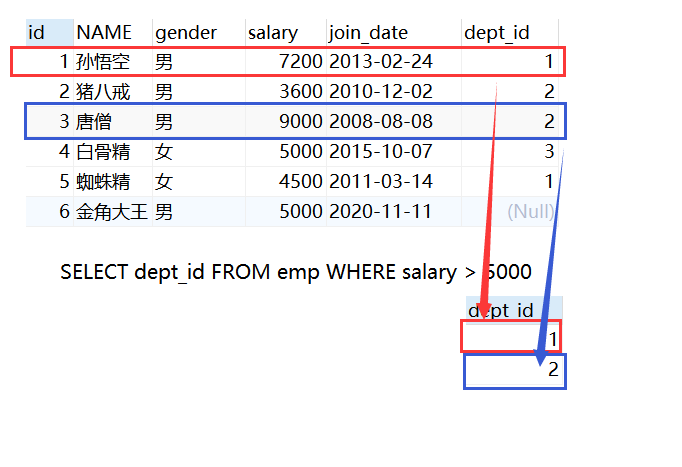
in、any、all

**示例**

**1、查询工资大于5000的员工，来自于哪些部门的名字**

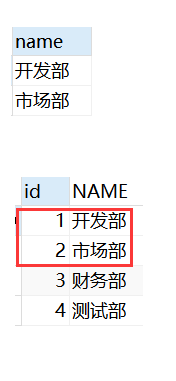
先查询大于5000的员工所在的部门id

SELECT dept\_id FROM emp WHERE salary > 5000



再查询在这些部门id中部门的名字

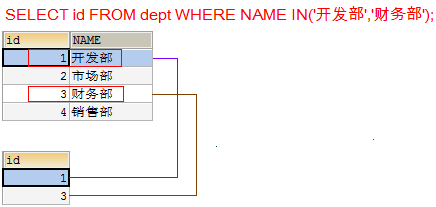
SELECT dept.name FROM dept WHERE dept.id IN (SELECT dept\_id FROM emp WHERE salary > 5000)



**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

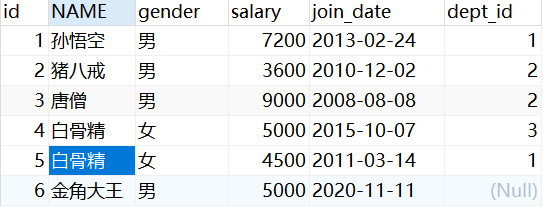
SELECT \* FROM emp WHERE dept\_id IN (SELECT id FROM dept WHERE NAME IN('开发部','财务部'))



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

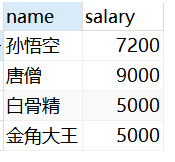


select salary from emp where name='白骨精'



**any表示任意值**

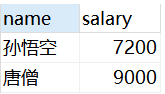
select name,salary from emp where salary >any(select salary from emp where name='白骨精')



>any表示比最小的大就可以(大于4500)

**all表示全部值**

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

子查询结果是单列，在WHERE后面作为条件 子查询结果

**子查询结果是多列，肯定在FROM后面作为表**

**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 。

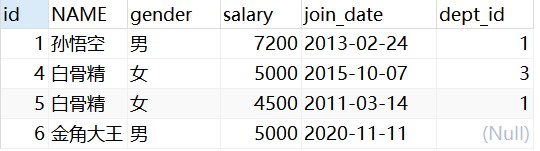
**子查询作为表需要取别名，否则这张表无法访问表中的字段。**

**示例**

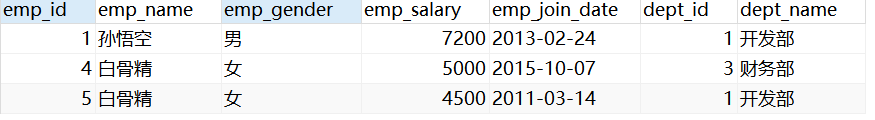
**查询出2011年以后入职的员工信息，包括部门名称**

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

## 五、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

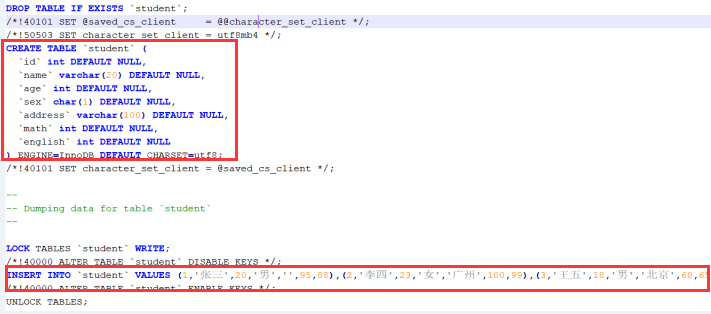
**示例**

备份mydb1数据库中的数据

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

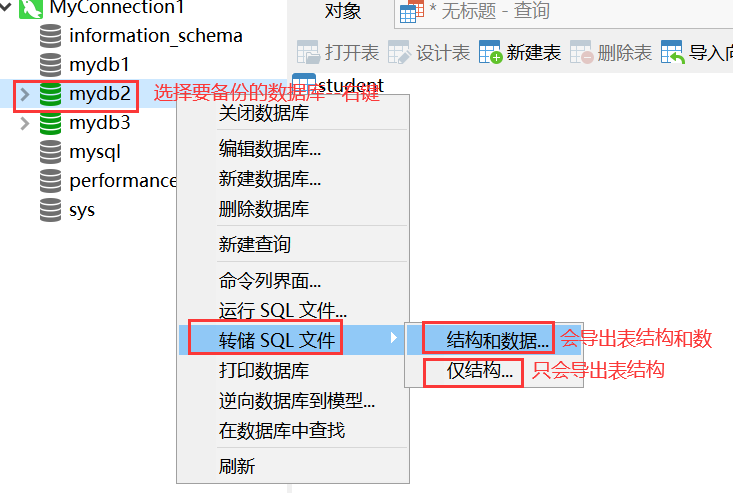






sql文件中将会包含相应的建表语句和插入语句

#### 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

**语法**

SOURCE 导入文件的路径;

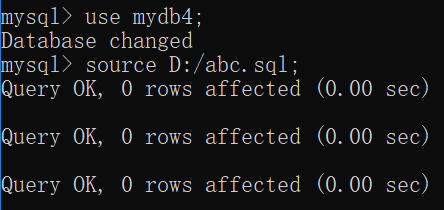
**示例**

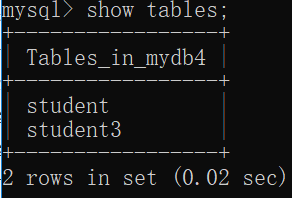
注意：还原时要先登录到MySQL中

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

