MySQL表的约束与数据库设计

# 回顾

## MySQL管理数据库

* 查看所有数据库

show databases;

* 创建数据库

create database 数据库名;

* 删除数据库

drop database 数据库名;

## 表的管理表

* 查看所有表

show tables;

* 创建表(student(id,name,age))

create table student (

id int,

name varchar(20),

age int

)

## 查看一个表结构：

* 以sql格式返回

show create table 表名;

* 以表格格式返回

desc 表名;

## 删除表

drop table 表名;

## 管理数据：数据增删改

* 插入数据

insert into 表名 values (字段列表)

* 修改数据

update 表名 set 字段名=值 where 条件

* 删除数据

1. 删除表中的所有数据

delete from 表名

2. 删除所有数据

truncate 表名

## WHERE子句中经常使用的运算符

|  |  |  |
| --- | --- | --- |
| 运算符类型 | 运算符 | 说明 |
| 比较运算符 | between…and | 在一个指定的范围之类，包头又包尾 |
| in | 查询集合中的元素，与集合中的元素相等 |
| like | 模糊查询  \_ 匹配1个  % 匹配多个 |
| is null | 为空，判断是否为空 |

# 学习目标

1. 能够使用SQL语句进行排序
2. 能够使用聚合函数
3. 能够使用SQL语句进行分组查询
4. 能够完成数据的备份和恢复
5. 能够使用SQL语句添加主键、外键、唯一、非空约束
6. 能够说出多表之间的关系及其建表原则
7. 能够理解三大范式

# DQL查询语句

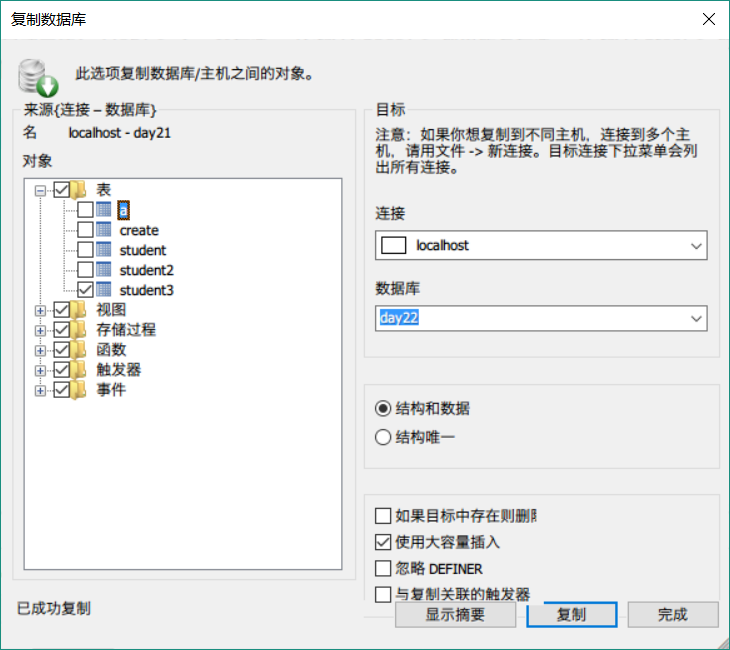
## 排序

对指定的一个或多个字段进行排序

|  |
| --- |
| 通过ORDER BY子句，可以将查询出的结果进行排序(排序只是显示方式，不会影响数据库中数据的顺序) |
| SELECT 字段名 FROM 表名 WHERE 字段=值 ORDER BY 字段名 [ASC|DESC]; |
| ASC: 默认，升序 |
| DESC: 降序排序 |

Order by出现在select语句的最后面

准备数据

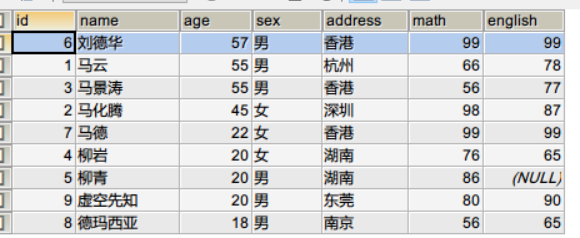


### 单列排序

* 什么是单列排序：对一列进行排序

-- 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

* 什么是组合排序?

同时对多列进行排序，先按第1列排序，如果第1列数据相同，则按第2列排序，依次类推。每个字段既可以使用asc，也可以使用desc。

组合排序的语法：

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

-- 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;

## 聚合函数

之前我们做的查询都是横向查询，它们都是根据条件一行一行的进行判断，而使用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

|  |  |
| --- | --- |
| SQL中的聚合函数 | 作用 |
| count(字段名) | 统计指定列记录数，记录为NULL的不统计 |
| sum(字段名) | 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(字段名) | 计算指定列的最大值 |
| min(字段名) | 计算指定列的最小值 |
| avg(字段名) | 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 |

### 语法：

|  |
| --- |
| SELECT 聚合函数(列名) FROM 表名; |

-- 查询学生总数

select count(id) from student3;

select count(english) from student3;

select count(\*) from student3;

我们发现对于NULL的记录不会统计，建议如果统计个数则不要使用有可能为null的列，但如果需要把NULL也统计进去呢？

|  |  |
| --- | --- |
| IFNULL(列名，默认值) | 参数1： 指定表中的某一列  参数2： 如果这一列有值，返回这列的值。如果这一列为NULL，就使用这个默认值 |

-- 查询english字段，如果为null，则使用0代替

select count(ifnull(english,0)) from student3;

我们可以利用IFNULL()函数，如果记录为NULL，给个默认值，这样统计的数据就不会遗漏

-- 查询年龄大于40的总数

select \* from student3 where age > 40;

select count(\*) from student3 where age > 40;

-- 查询数学成绩总分

select sum(math) from student3;

-- 查询数学成绩平均分

select avg(math) from student3;

-- 查询数学成绩最高分

select max(math) from student3;

-- 查询数学成绩最低分

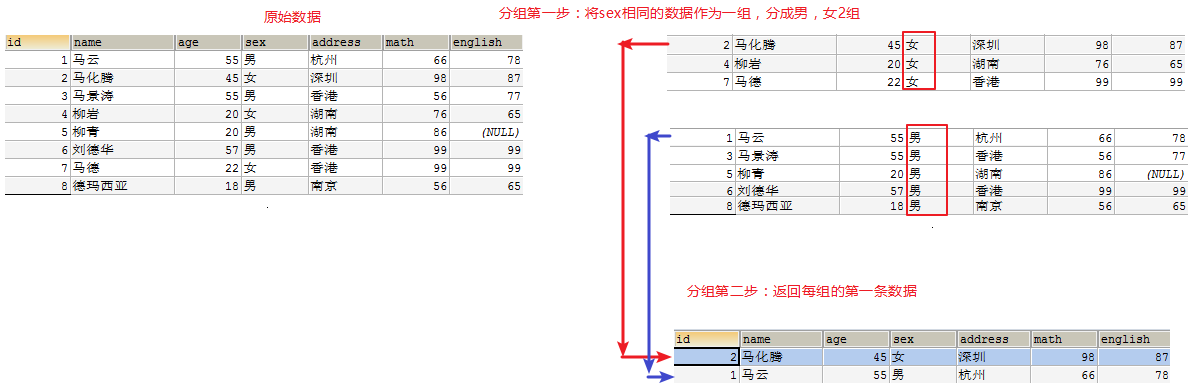
select min(math) from student3;

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

* GROUP BY怎么分组的？

将分组字段结果中相同内容作为一组，如按性别将学生分成2组。



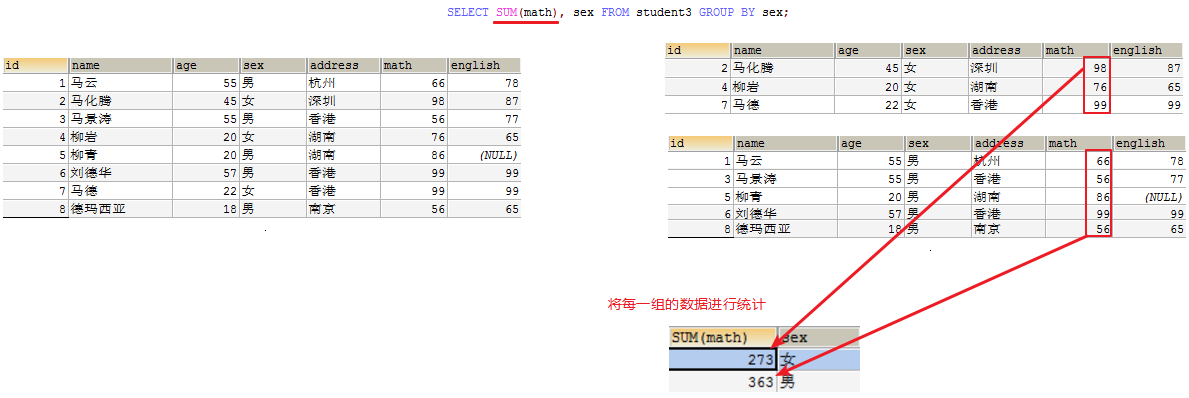
GROUP BY将分组字段结果中相同内容作为一组，并且返回每组的第一条数据，所以单独分组没什么用处。分组的目的就是为了统计，一般分组会跟聚合函数一起使用。

-- 按性别进行分组，求男生和女生数学的总成绩

select sex,sum(math) from student3 group by sex;

* 效果如下：

fig:

实际上是将每组的math进行求和,返回每组统计的结果 

* + 注意：当我们使用某个字段分组,在查询的时候也需要将这个字段查询出来,否则看不到数据属于哪组的
* 查询男女各多少人

1. 查询所有数据,按性别分组。
2. 统计每组人数

select count(\*), sex from student3 group by sex;



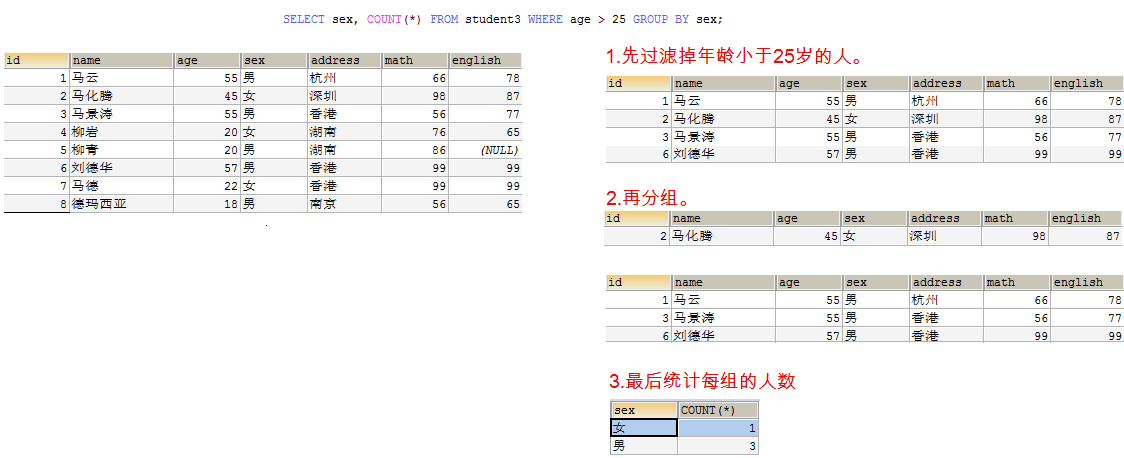
* 查询年龄大于25岁的人,按性别分组,统计每组的人数

1. 先过滤掉年龄小于25岁的人。
2. 再分组。
3. 最后统计每组的人数

-- 查询年龄大于25岁的人,按性别分组,统计每组的人数

select \* from student3 where age > 25;

select count(\*) ,sex from student3 where age > 25 group by sex;



* 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  
  以下代码是否正确？

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

* 正确写法：

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`这组数据显示出来



### having与where的区别

|  |  |
| --- | --- |
| 子名 | 作用 |
| where 子句 | 1. 对查询结果进行分组前，将不符合where条件的行去掉，即在分组之前过滤数据，即先过滤再分组。 2. where后面不可以使用聚合函数 |
| having子句 | 1. having 子句的作用是筛选满足条件的组，即在分组之后过滤数据，即先分组再过滤。 2. having后面可以使用聚合函数 |

## limit分页语句

### 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);

### limit的作用：

LIMIT是限制的意思，所以LIMIT的作用就是限制查询记录的条数。

|  |
| --- |
| 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]; |

### LIMIT语法格式：

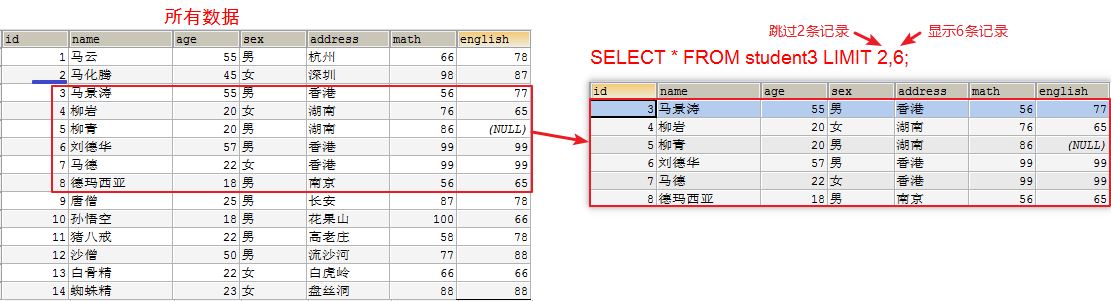
|  |
| --- |
| LIMIT offset,length; |
| offset：起始的行数，从0开始，可以省略 |
| length： 得到记录的行数 |

-- 查询学生表中数据，从第3条开始显示，显示6条。

select \* from student3 limit 2,6;

select \* from student3 limit 3;

select \* from student3 limit 0, 3;



### 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。 假设我们一每页显示5条记录的方式来分页。

-- 每页显示5条  
-- 第一页： LIMIT 0,5; 跳过0条，显示5条

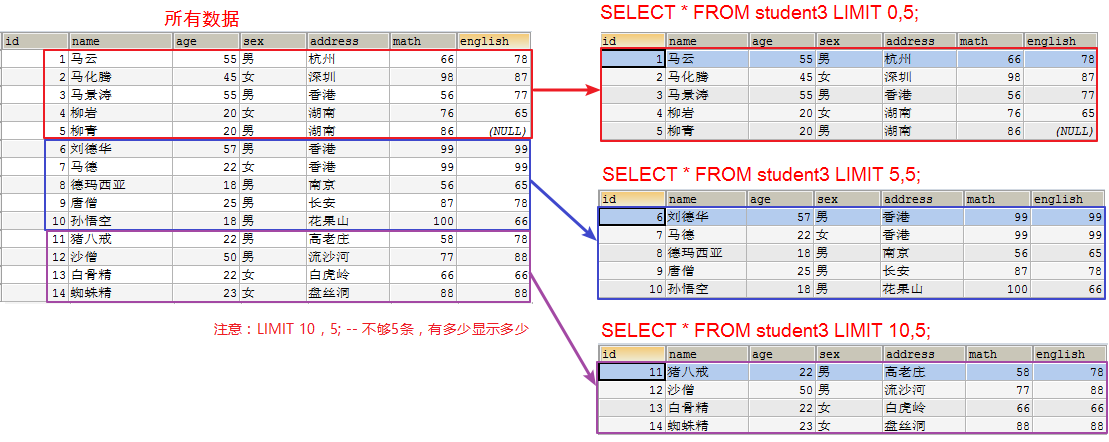
select \* from student3 limit 0,5;

-- 第二页： LIMIT 5,5; 跳过5条，显示5条

select \* from student3 limit 5,5;

-- 第三页： LIMIT 10,5; 跳过10条，显示5条

select \* from student3 limit 10,5;



* 如果第一个参数是0可以省略写，最后如果不够5条，有多少显示多少

# 数据库备份

## 备份的应用场景

在服务器进行数据传输、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。

## 备份与还原的语句

### 备份格式：

运行在DOS命令下

|  |
| --- |
| mysqldump -u用户名 -p密码 数据库 > 文件的路径 |

### 还原格式：

运行在登录mysql以后命令行一下，在运行source之前，要先选中某个数据库

|  |
| --- |
| source导入文件的路径; |

### 备份操作：

* 备份day21数据库中的数据到d:\day21.sql文件中

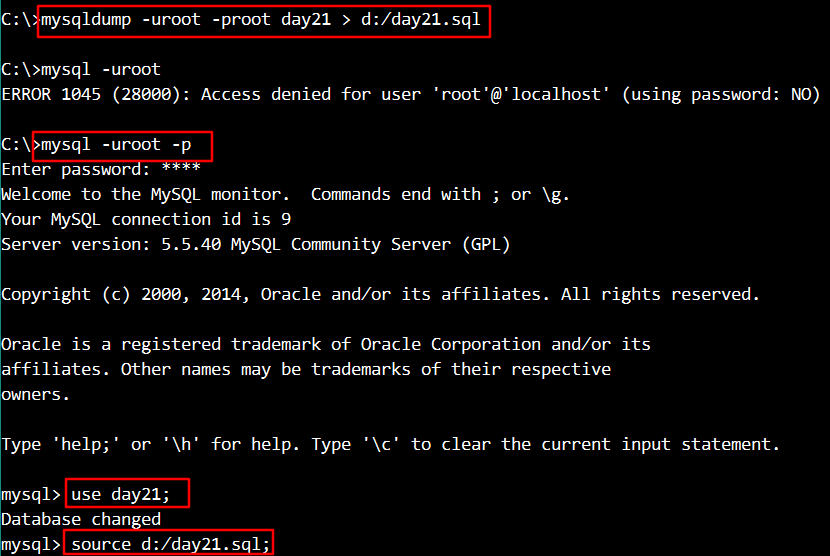
mysqldump -uroot -proot day21 > d:/day21.sql

导出结果：数据库中的所有表和数据都会导出成SQL语句

### 还原操作

* 还原day21数据库中的数据，注意：还原的时候需要先登录MySQL,并选中对应的数据库。

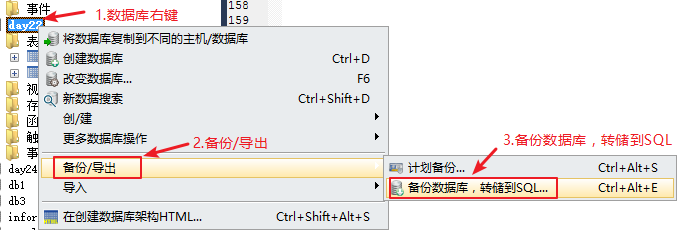
1. 删除day21数据库中的所有表
2. 登录MySQL
3. 选中数据库
4. 使用SOURCE命令还原数据
5. 查看还原结果

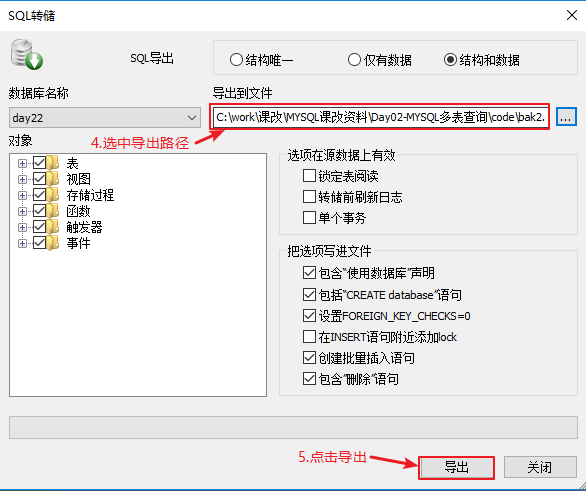


## 图形化界面备份与还原

### 备份数据库中的数据

1. 选中数据库，右键 ”备份/导出”
2. 指定导出路径，保存成.sql文件即可。

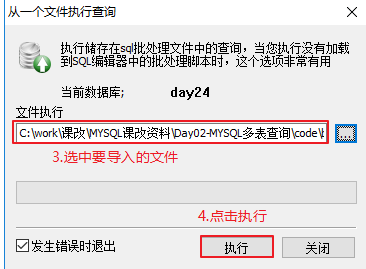




### 还原数据库中的数据

1. 删除数据库
2. 数据库列表区域右键“执行SQL脚本”， 指定要执行的SQL文件，执行即可





# 数据库约束

## 数据库约束的概述

### 约束的作用：

对表中的数据进行限制，保证数据的正确性、有效性和完整性。理想的是数据表中所有的数据是准确的，所以我们必须对添加进入表中的数据进行约束。如果违反约束条件，数据添加失败。

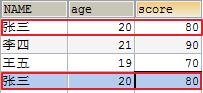
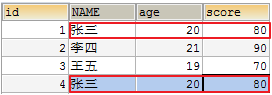
### 约束种类：

|  |  |
| --- | --- |
| 约束名 | 约束关键字 |
| 主键 | primary key |
| 唯一 | unique |
| 非空 | not null |
| 默认 | default |
| 外键 | foreign key |
| 检查 | 在mysql中不支持 |

## 主键约束

### 主键的作用

代表每条记录的标识，它必须是唯一，而且是非空的。主键可以是任何类型，一般使用int或字符串。主键是否连接没有关系。

### 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。

最好是在创建表的时候创建约束

### 创建主键

* 主键关键字： primary key
* 主键的特点：

1. 唯一
2. 非空

* 创建主键方式：

1. **在创建表的时候给字段添加主键**

|  |
| --- |
| 字段名 字段类型 PRIMARY KEY |

1. **在已有表中添加主键**

|  |
| --- |
| ALTER TABLE 表名 ADD PRIMARY KEY(字段名); |

-- 创建表学生表st1, 包含字段(id, name, age)将id做为主键

create table st1 (

id int primary key,

name varchar(20),

age int

);

desc st1;

select \* from st1;

-- 插入重复的主键值

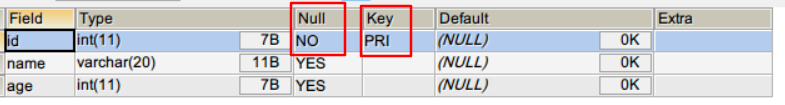
insert into st1 values (1,'张三',48);

-- Duplicate 重复的 entry记录 '1' for key 'PRIMARY'

insert into st1 values (1,'李白',40);

-- 插入NULL的主键值 Column 'id' cannot be null

insert into st1 values (NULL,'张三',48);



### 删除主键

-- 删除st1表的主键

alter table st1 drop primary key;

-- 添加主键

alter table st1 add primary key(id);

### 主键自增

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。只能用在主键。

|  |
| --- |
| AUTO\_INCREMENT 表示自动增长(字段类型必须是整数类型) |

-- 创建学生表st2, 包含字段(id, name, age)将id做为主键并自动增长

create table st2 (

id int primary key auto\_increment,

name varchar(20),

age int

);

desc st2;

-- 插入数据，可以省略id这列

select \* from st2;

-- Column count doesn't match value count at row 1

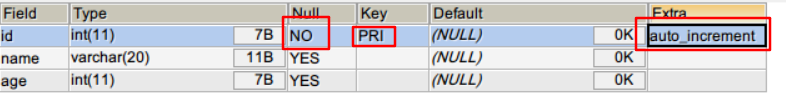
insert into st2 values ('张三',20);

-- 指定列名

insert into st2 (name,age) values ('张三疯',23);

-- 主键因为自动增长

insert into st2 values (null, '张四疯',23);





### 修改自增长的默认值起始值

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



* 创建表时指定起始值

|  |
| --- |
| CREATE TABLE 表名(  列名 int primary key AUTO\_INCREMENT  ) AUTO\_INCREMENT=起始值; |

* 创建好以后修改起始值

|  |
| --- |
| ALTER TABLE 表名 AUTO\_INCREMENT=起始值; |

-- 指定起始数为1000

create table st2 (

id int primary key auto\_increment,

name varchar(20),

age int

) auto\_increment = 1000;

-- 指定列名

insert into st2 (name,age) values ('张三疯',23);

-- 主键因为自动增长

insert into st2 values (null, '张四疯',23);

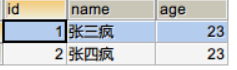
select \* from st2;

-- 创建好以后修改

alter table st2 auto\_increment = 2000;

### DELETE和TRUNCATE对自增长的影响

* DELETE：不会影响之前的自动增长编号
* TRUNCATE：从1开始重新计数



## 唯一约束

什么是唯一约束： 这一列中不能出现重复的数据

### 唯一约束的基本格式

|  |
| --- |
| 字段名 字段类型 UNIQUE |

### 实现唯一约束



-- 创建学生表st3, 包含字段(id, name),name这一列设置唯一约束,不能出现同名的学生

create table st3 (

id int primary key,

name varchar(20) unique

)

-- 添加一个同名的学生

select \* from st3;

insert into st3 values(1, '张三');

-- Duplicate entry '张三' for key 'name'

insert into st3 values(2, '张三');

insert into st3 values(2, '李四');

-- 重复插入多个null会怎样？ null表示没有数据

insert into st3 values(3, NULL);

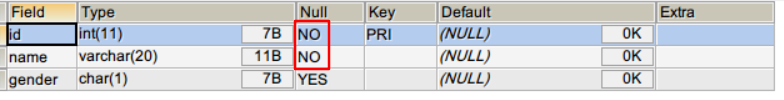
insert into st3 values(4, NULL);

## 非空约束

* 什么是非空约束：这列不能为NULL，必须要有值

### 非空约束的基本语法格式

|  |
| --- |
| 字段名 字段类型 NOT NULL |



-- 创建表学生表st4, 包含字段(id,name,gender)其中name不能为NULL

create table st4 (

id int primary key,

name varchar(20) not null,

gender char(1)

);

desc st4;

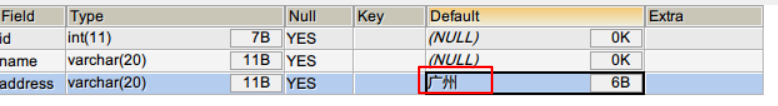
-- 添加一条记录其中姓名不赋值, Column 'name' cannot be null

insert into st4 values(1,null,'男');

### 默认值

什么是默认值：如果某一列，没有输入值，则使用默认值来代替。

|  |
| --- |
| 字段名 字段类型 DEFAULT 默认值 |



-- 创建一个学生表 st5，包含字段(id,name,address)， 地址默认值是广州

create table st5(

id int,

name varchar(20),

address varchar(20) default '广州'

);

desc st5;

-- 添加一条记录,使用默认地址

insert into st5 (id,name) values (1,'潘金莲');

select \* from st5;

-- 不行

insert into st5 values (1,'潘金莲', null);

insert into st5 values (1,'潘金莲', default);

-- 添加一条记录,不使用默认地址

insert into st5 values (1,'潘金莲', '东莞');

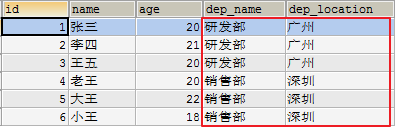
## 外键约束

### 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5条数据

CREATE TABLE emp (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(30),  
 age INT,  
 dep\_name VARCHAR(30), -- 部门名字  
 dep\_location VARCHAR(30) -- 部门所在地  
);  
  
-- 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');

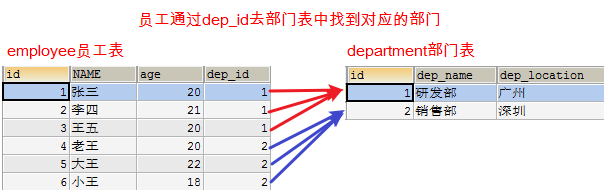
* 以上数据表的缺点:
  1. 如果员工有1000条记录，而部门只有10个部门。大量重复的部门数据在里面。
  2. 会出现数据的错误，输入：李四，研发部，深圳
  3. 会出现部门丢失的问题：如果某个部门只有一个人：李白，小卖部。如果删除李白，会导致部门也没有了。



### 解决方案:

分成两张表，一张是员工表，一张是部门表。每张表都应该有主键

在员工表中多添加一列，dep\_id,与部门表中主键有一个对应的关系，说明当前这个员工在哪个部门。这一列称为外键。



-- 创建部门表(id,dep\_name,dep\_location)

-- 多个单词用下划线分隔

create table department(

id int primary key auto\_increment,

dep\_name varchar(20),

dep\_location varchar(20)

);

-- 创建员工表(id,name,age,dep\_id)

create table employee(

id int primary key auto\_increment,

name varchar(20),

age int,

dep\_id int -- 外键

);

-- 添加2个部门

insert into department values (null, '研发部', '广州');

insert into department values (null, '公关部', '东莞');

select \* from department;

-- 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);

INSERT INTO employee (NAME, age, dep\_id) VALUES ('李四', 21, 1);

INSERT INTO employee (NAME, age, dep\_id) VALUES ('王五', 20, 1);

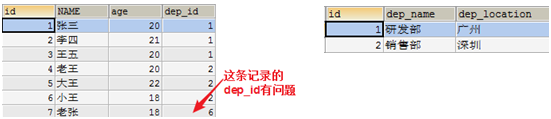
INSERT INTO employee (NAME, age, dep\_id) VALUES ('老王', 20, 2);

INSERT INTO employee (NAME, age, dep\_id) VALUES ('大王', 22, 2);

INSERT INTO employee (NAME, age, dep\_id) VALUES ('凤姐', 18, 2);

select \* from employee;

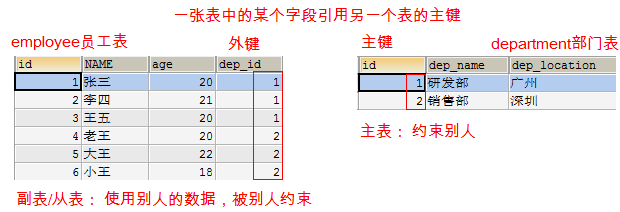
* **问题**：当我们在employee的dep\_id里面输入不存在的部门,数据依然可以添加.但是并没有对应的部门，实际应用中不能出现这种情况。employee的dep\_id中的数据只能是department表中存在的id



* **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

### 什么是外键约束

* 什么是外键： 外键出现在从表，用来关联主表列，必须与主表中的主键类型要相同。
* 主表： 部门表。部门对员工是一对多的关系，主表是1方，从表是多方。
* 从表： 有外键的这张表是从表，员工表。



### 创建约束的语法

* 新建表时增加外键：

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

* 已有表增加外键：

|  |
| --- |
| ALTER TABLE 从表 ADD [CONSTRAINT] [外键约束名称] FOREIGN KEY (外键字段名) REFERENCES 主表(主键字段名); |

* 具体操作：

1. 删除副表/从表 employee
2. 创建从表 employee 并添加外键约束emp\_depid\_fk
3. 正常添加数据
4. 部门错误的数据添加失败

-- 1) 删除副表/从表 employee

drop table employee;

-- 2) 创建从表 employee 并添加外键约束emp\_depid\_fk

create table employee(

id int primary key auto\_increment,

name varchar(20),

age int,

dep\_id int, -- 外键

constraint emp\_depid\_fk foreign key (dep\_id) references department(id)

);

select \* from employee;

-- 3) 正常添加数据

INSERT INTO employee (NAME, age, dep\_id) VALUES ('大王', 22, 2);

INSERT INTO employee (NAME, age, dep\_id) VALUES ('凤姐', 18, 2);

-- 4) 部门错误的数据添加失败

-- Cannot add or update a child row: a foreign key constraint fails

INSERT INTO employee (NAME, age, dep\_id) VALUES ('关羽', 30, 3);

### 删除外键

|  |
| --- |
| ALTER TABLE 从表 drop foreign key 外键名称; |

-- 删除employee表的emp\_depid\_fk外键

alter table employee drop foreign key emp\_depid\_fk;

-- 在employee表情存在的情况下添加外键

alter table employee add constraint emp\_depid\_fk foreign key (dep\_id) references department(id)

### 外键的级联

-- 要把部门表中的id值2，改成5，能不能直接更新呢

select \* from department;

select \* from employee;

-- Cannot delete or update a parent row

update department set id = 5 where id = 2;

--要删除部门id等于1的部门, 能不能直接删除呢

-- Cannot delete or update a parent row

delete from department where id=2;

* 什么是级联操作：

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

|  |  |
| --- | --- |
| 级联操作语法 | 描述 |
| ON UPDATE CASCADE | 级联更新，更新主表中的主键，从表中的外键也同时更新 |
| ON DELETE CASCADE | 级联删除，删除主表中的一行，从表中相应的外键行也全部删除 |

drop table department; -- 如果有从表，主表不能删除

drop table employee;

-- 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(20),

age int,

dep\_id int, -- 外键

constraint emp\_depid\_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 ('凤姐', 18, 2);

-- 把部门表中id等于1的部门改成id等于10

select \* from employee;

select \* from department;

update department set id=10 where id=1;

-- 删除部门号是2的部门

delete from department where id=2;

## 数据约束小结

|  |  |  |
| --- | --- | --- |
| 约束名 | 关键字 | 说明 |
| 主键 | primary key | 唯一标识一条记录 |
| 默认 | default | 指定默认值 |
| 非空 | not null | 列不能空 |
| 唯一 | unique | 列不能有重复值 |
| 外键 | foreign key | 从表中外键与主表中主键要匹配 |

# 表与表之间的关系

## 表关系的概念

现实生活中，实体与实体之间肯定是有关系的，比如：老公和老婆，部门和员工，老师和学生等。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！

|  |  |
| --- | --- |
| 表与表之间的三种关系 | 说明 |
| 一对多 | 最常用的关系，反之也称为多对一 |
| 多对多 | 通过一张中间表来实现这种关系 |
| 一对一 | 一个表中一条记录对应另一张表中一条记录 |

## 一对多

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

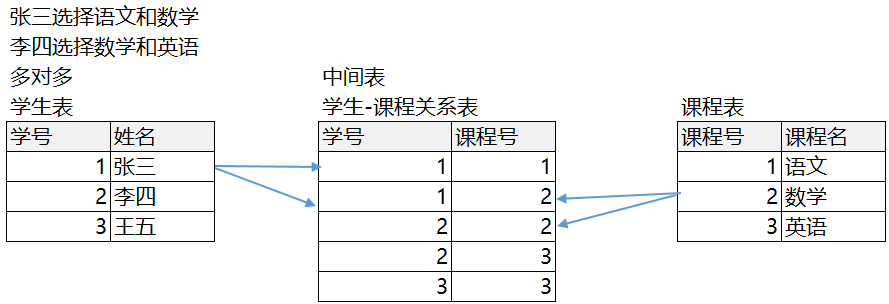
一对多建表原则: 在从表(多方)创建一个字段,字段作为外键指向主表(一方)的主键



## 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，这两个字段分别作为外键指向各自一方的主键。 其实就是两个一对多加到一起变成了多对多的关系。

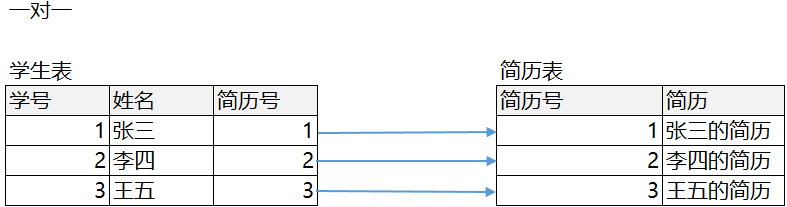


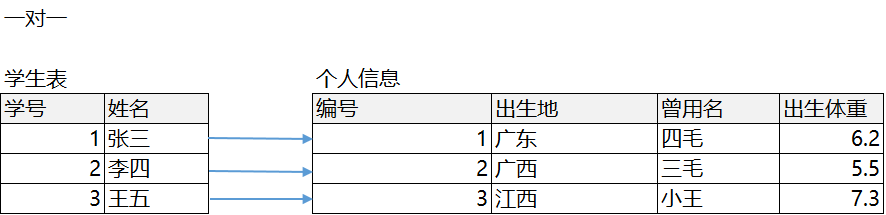
## 一对一

一对一（1:1） 在实际的开发中应用不多.因为一对一可以创建成一张表。 如：公民表 🡪 护照表

两种建表原则：

|  |  |
| --- | --- |
| 一对一的建表原则 | 说明 |
| 外键唯一 | 主表的主键和从表的外键（唯一），形成主外键关系，外键唯一UNIQUE |
| 外键是主键 | 从表的主键同时又是外键，主表的主键列与从表的主键列是一一对应的关系。 |





## 一对多关系案例

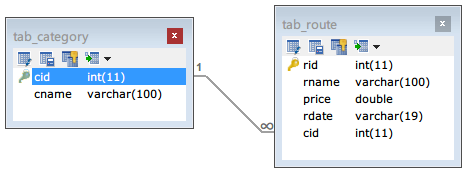
### 需求：一个旅游分类中有多个旅游线路

* 界面



* 表与表的关系





### 具体操作：

-- 创建旅游线路分类表tab\_category

/\*

cid旅游线路分类主键，自动增长

cname旅游线路分类名称非空，唯一，字符串100

\*/

create table tab\_category (

cid int primary key auto\_increment,

cname varchar(100) not null unique

);

-- 添加旅游线路分类数据：('周边游'), ('出境游'), ('国内游'), ('港澳游')

insert into tab\_category (cname) values ('周边游'), ('出境游'), ('国内游'), ('港澳游');

select \* from tab\_category;

-- 创建旅游线路表tab\_route

/\*

rid旅游线路主键，自动增长

rname旅游线路名称非空，唯一，字符串100

price价格

rdate 上架时间，日期类型

cid 外键，所属分类

\*/

drop table tab\_route;

create table tab\_route (

rid int primary key auto\_increment,

rname varchar(100) not null unique,

price double,

rdate date,

cid int,

foreign key (cid) references tab\_category(cid)

)

-- 添加旅游线路数据

INSERT INTO tab\_route VALUES

(NULL, '【厦门+鼓浪屿+南普陀寺+曾厝垵 高铁3天 惠贵团】尝味友鸭面线 住1晚鼓浪屿', 1499, '2018-01-27', 1),

(NULL, '【浪漫桂林 阳朔西街高铁3天纯玩 高级团】城徽象鼻山 兴坪漓江 西山公园', 699, '2018-02-22', 3),

(NULL, '【爆款￥1699秒杀】泰国 曼谷 芭堤雅 金沙岛 杜拉拉水上市场 双飞六天【含送签费 泰风情 广州往返 特价团】', 1699, '2018-01-27', 2),

(NULL, '【经典•狮航 ￥2399秒杀】巴厘岛双飞五天 抵玩【广州往返 特价团】', 2399, '2017-12-23', 2),

(NULL, '香港迪士尼乐园自由行2天【永东跨境巴士广东至迪士尼去程交通+迪士尼一日门票+香港如心海景酒店暨会议中心标准房1晚住宿】', 799, '2018-04-10', 4);

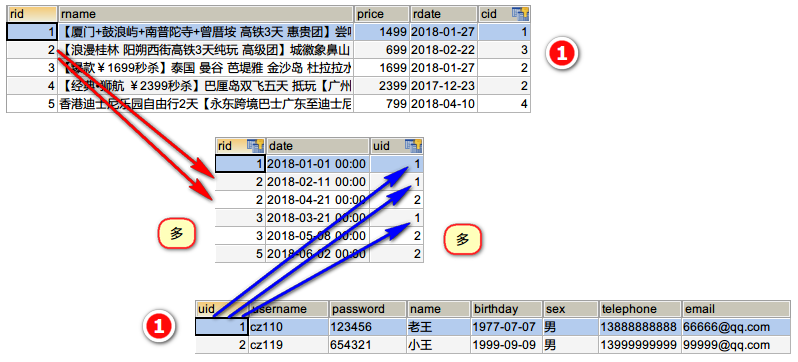
select \* from tab\_route;

select \* from tab\_category;

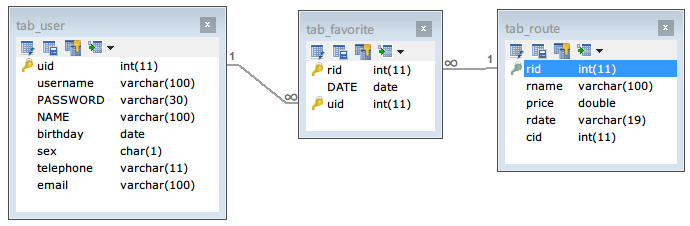
## 多对多关系案例

### 需求：一个用户收藏多个线路，一个线路被多个用户收藏





对于多对多的关系我们需要增加一张中间表来维护他们之间个关系



### 具体操作：

/\*

创建用户表tab\_user

uid用户主键，自增长

username用户名长度100，唯一，非空

password密码长度30，非空

name真实姓名长度100

birthday生日

sex性别，定长字符串1

telephone手机号，字符串11

email邮箱，字符串长度100

\*/

create table tab\_user (

uid int primary key auto\_increment,

username varchar(100) unique not null,

password varchar(30) not null,

name varchar(100),

birthday date,

sex char(1) default '男',

telephone varchar(11),

email varchar(100)

)

-- 添加用户数据

INSERT INTO tab\_user VALUES

(NULL, 'cz110', 123456, '老王', '1977-07-07', '男', '13888888888', '66666@qq.com'),

(NULL, 'cz119', 654321, '小王', '1999-09-09', '男', '13999999999', '99999@qq.com');

select \* from tab\_user;

/\*

创建收藏表tab\_favorite

rid 旅游线路id，外键

date 收藏时间

uid用户id，外键

rid和uid不能重复，设置复合主键，同一个用户不能收藏同一个线路两次

\*/

drop table tab\_favorite;

create table tab\_favorite(

rid int,

`date` datetime,

uid int,

primary key(rid,uid), -- 联合主键

foreign key(rid) references tab\_route(rid), -- 外键

foreign key(uid) references tab\_user(uid) -- 外键

)

-- 增加收藏表数据

INSERT INTO tab\_favorite VALUES

(1, '2018-01-01', 1), -- 老王选择厦门

(2, '2018-02-11', 1), -- 老王选择桂林

(3, '2018-03-21', 1), -- 老王选择泰国

(2, '2018-04-21', 2), -- 小王选择桂林

(3, '2018-05-08', 2), -- 小王选择泰国

(5, '2018-06-02', 2); -- 小王选择迪士尼

-- Duplicate entry '1-1' for key 'PRIMARY'

INSERT INTO tab\_favorite VALUES

(1, '2018-01-01', 1);

## 表与表之间的关系小结

|  |  |
| --- | --- |
| 表与表的关系 | 关系的维护 |
| 一对多 | 通过外键来维护 |
| 多对多 | 通过中间，将两个1对多变成多对多 |
| 一对一 | 1. 特殊的1对多，多方使用唯一约束 2. 从表的主键， 同时又是外键 |

1. 数据库设计
   1. 数据规范化
      1. 什么是范式：

好的数据库设计对数据的存储性能和后期的程序开发，都会产生重要的影响。建立科学的，规范的数据库就需要满足一些规则来优化数据的设计和存储，这些规则就称为范式。

* + 1. 三大范式：

目前关系数据库有六种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、巴斯-科德范式（BCNF）、第四范式(4NF）和第五范式（5NF，又称完美范式）。

满足最低要求的范式是第一范式（1NF）。在第一范式的基础上进一步满足更多规范要求的称为第二范式（2NF），其余范式以次类推。数据库只需满足第三范式(3NF）就行了。

* 1. 第一范式
     1. 概念：

数据库表的每一列都是不可分割的原子数据项，不能是集合、数组等非原子数据项。即表中的某个列有多个值时，必须拆分为不同的列。简而言之，第一范式每一列不可再拆分，称为原子性。

* + 1. 班级表

|  |  |  |
| --- | --- | --- |
| 学号 | 姓名 | 班级 |
| 1 | 张三 | 一年三级 |
| 2 | 李四 | 二年二级 |
| 3 | 王五 | 一年二级 |

试卷答案：A,B,C,A,D

收货地址：xx省xx市

* 解决方案：

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 年级 | 班号 |

* 1. 第二范式
     1. 概念：

在满足第一范式的前提下，表中的每一个字段都完全依赖于主键。不存在局部依赖

所谓完全依赖是指不能存在仅依赖主键一部分的列。简而言之，第二范式就是在第一范式的基础上所有列完全依赖于主键列。当存在一个复合主键包含多个列的时候，才会发生不符合第二范式的情况。比如有一个主键有两个列，不能存在这样的属性，它只依赖于其中一个列，这就是不符合第二范式。

第二范式的特点：

* + 一张表只描述一件事情。
  + 表中的每一列都完全依赖于主键
    1. 示例：

1) 借书证表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生证号 | 学生证名称 | 学生证办理时间 | 借书证号 | 借书证名称 | 借书证办理时间 |

2) 解决方案：

|  |  |  |
| --- | --- | --- |
| 学生证号 | 学生证名称 | 学生证办理时间 |

|  |  |  |
| --- | --- | --- |
| 借书证号 | 借书证名称 | 借书证办理时间 |

* 1. 第三范式：
     1. 概念：

在满足第二范式的前提下，表中的每一列都直接依赖于主键，而不是通过其它的列来间接依赖于主键。

简而言之，第三范式就是所有列不依赖于其它非主键列，也就是在满足2NF的基础上，任何非主列不得传递依赖于主键。所谓传递依赖，指的是如果存在"A → B → C"的决定关系，则C传递依赖于A。因此，满足第三范式的数据库表应该不存在如下依赖关系：主键列 → 非主键列x → 非主键列y

* + 1. 示例：学生信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学号 | 姓名 | 年龄 | 所在学院 | 学院地点 |

* 存在传递的决定关系：

学号 🡪 所在学院 🡪 学院地点

有传递依赖关系：

学院地点通过所在学院来依赖于学号

* 解决方案

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 年龄 | 学院ID |

|  |  |  |
| --- | --- | --- |
| 学院ID | 所在学院 | 学院地点 |

* + 1. 三大范式小结：

|  |  |
| --- | --- |
| 范式 | 特点 |
| 第一范式 | 原子性，表中的每一列都不能再拆分 |
| 第二范式 | 不产生局部依赖，表中每一列都完全依赖于主键。 |
| 第三范式 | 不产生传递依赖，表中每一列都直接依赖于主键，而不是间接依赖于主键。 |