



MySQL 表的约束与数据库设计

第1节 回顾

1.1 数据库入门

1.1.1 SQL 语句的分类：

- 1) DDL 数据定义语言
- 2) DML 数据操作语言
- 3) DQL 数据查询语言
- 4) DCL 数据控制语言

1.1.2 MySQL 管理数据库

- 查看所有数据库

```
show databases;
```

- 创建数据库

```
create database 库名;
```

- 查看数据库创建数据的语句：

```
show create database 库名;
```

- 删除数据库

```
drop database 库名;
```

1.2 表的管理

- 查看所有表

```
show tables;
```

- 创建表(student(id,name,age))

```
create table student(  
    id int,  
    name varchar(20),  
    age int  
)
```

1.3 查看表结构：

- 以 sql 格式返回

```
show create table 表名;
```

- 以表格格式返回

```
desc 表名;
```

1.4 删除表

```
drop table 表名;
```

1.5 管理数据：数据增删改

- 插入数据

```
insert into 表名 (列名) values (值);
```

- 修改数据

```
update 表名 set 列名=值 where 条件
```

- 删除数据

1. 删除表中的所有数据

```
delete from 表名 where 条件
```

2. 删除所有数据

```
truncate [table] 表名
```

1.6 查询数据

- 查询所有列

```
select * from 表名
```

- 查询时指定别名

```
as 可以省略
```

- 去除重复数据

```
distinct
```

1.7 条件查询

- 1) 显示在某一区间的值: 80~100 between 80 and 100
- 2) 多个条件中符合 1 个值: in
- 3) 模糊查询: like % 匹配多个字符 _ 匹配 1 个

第2节 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

第3节 DQL 查询语句

3.1 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: 升序，默认值

DESC: 降序

3.1.1 单列排序

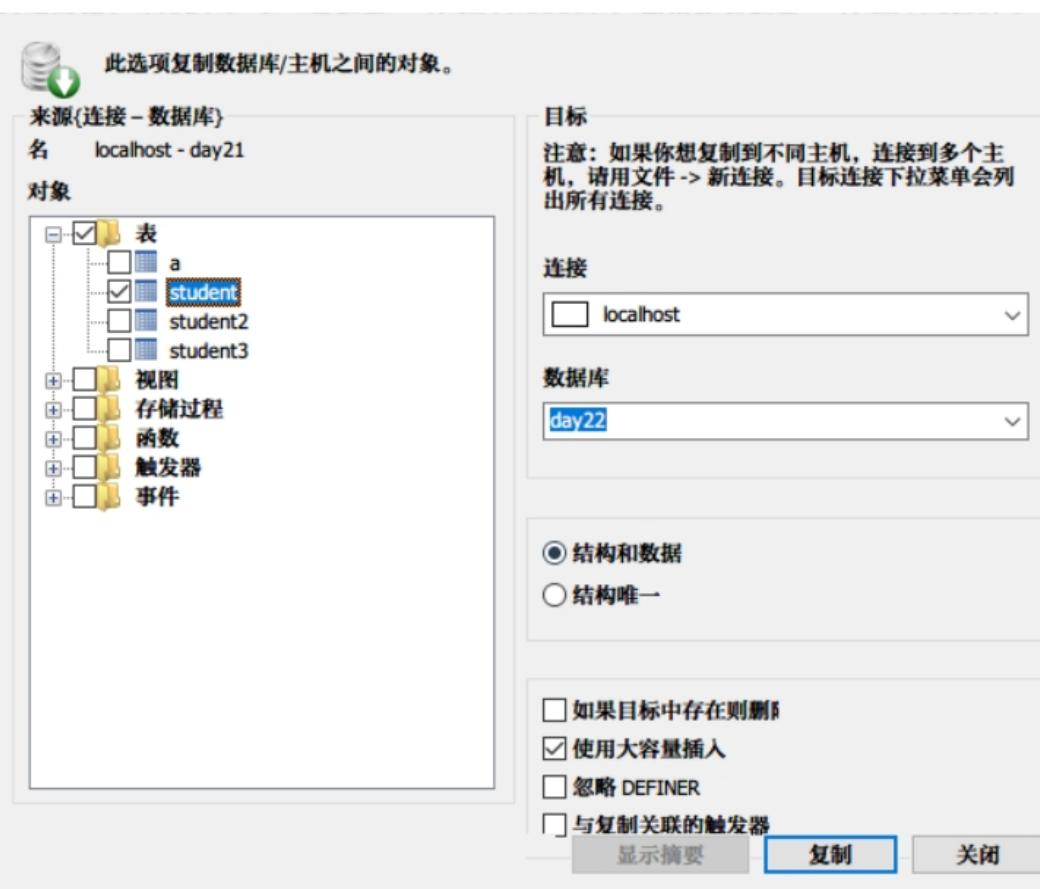
- 什么是单列排序：

只按某一个字段进行排序，单列排序。

3.1.2 实现不同数据库之间表的复制

复制数据库

X



-- 查询所有数据, 使用年龄降序排序

```
select * from student order by age desc;
```

3.1.3 组合排序

- 什么是组合排序?

同时对多个字段进行排序, 如果第 1 个字段相等, 则按第 2 个字段排序, 依次类推。

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升序排序

```
select * from student order by age desc, math asc;
```

3.2 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 NULL。

3.2.1 五个聚合函数

SQL 中的聚合函数	作用
max(列名)	求这一列的最大值
min(列名)	求这一列的最小值
avg(列名)	求这一列的平均值

count(列名)	统计这一列有多少条记录
sum(列名)	对这一列求总和

3.2.2 语法：

SELECT 聚合函数(列名) FROM 表名;

-- 查询学生总数

```
select count(id) as 总人数 from student;
```

```
select count(*) as 总人数 from student;
```

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)

如果列名不为空，返回这列的值。如果为 **NULL**，则返回默认值。

-- 查询 id 字段，如果为 null，则使用 0 代替

```
select ifnull(id,0) from student;
```

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

```
select count(ifnull(id,0)) from student;
```

-- 查询年龄大于 20 的总数

```
select count(*) from student where age>20;
```

-- 查询数学成绩总分

```
select sum(math) 总分 from student;
```

-- 查询数学成绩平均分

```
select avg(math) 平均分 from student;
```

-- 查询数学成绩最高分

```
select max(math) 最高分 from student;
```

-- 查询数学成绩最低分

```
select min(math) 最低分 from student;
```

3.3 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。

原始数据						
id	name	age	sex	address	math	english
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

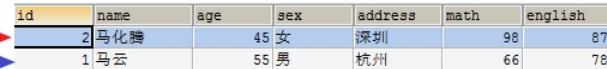
分组第一步：将sex相同的数据作为一组，分成男，女2组



2 马化腾	45 女	深圳	98	87
4 柳岩	20 女	湖南	76	65
7 马德	22 女	香港	99	99

1 马云	55 男	杭州	66	78
3 马景涛	55 男	香港	56	77
5 柳青	20 男	湖南	86	(NULL)
6 刘德华	57 男	香港	99	99
8 德玛西亚	18 男	南京	56	65

分组第二步：返回每组的第一条数据



id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87
1	马云	55	男	杭州	66	78

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平均分

```
select sex, avg(math) from student3 group by sex;
```

- 效果如下：

sex	avg(math)
女	91.0000
男	72.6000

实际上是将每组的 math 求了平均,返回每组统计的结果

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

id	name	age	sex	address	math	english
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



id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87
4	柳岩	20	女	湖南	76	65
7	马德	22	女	香港	99	99

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
8	德玛西亚	18	男	南京	56	65

将每一组的数据进行统计

SUM(math)	sex
273	女
363	男

◇ 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 sex, count(*) from student3 group by sex;
```

id	<th>age</th> <th>sex</th> <th>address</th> <th>math</th> <th>english</th>	age	sex	address	math	english
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

sex
女
男

相同的[数据作为一组](#)

- 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

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

id	<th>age</th> <th>sex</th> <th>address</th> <th>math</th> <th>english</th>	age	sex	address	math	english
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

1.先过滤掉年龄小于25岁的人。

id	<th>age</th> <th>sex</th> <th>address</th> <th>math</th> <th>english</th>	age	sex	address	math	english
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
6	刘德华	57	男	香港	99	99

2.再分组。

id	<th>age</th> <th>sex</th> <th>address</th> <th>math</th> <th>english</th>	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87

id	<th>age</th> <th>sex</th> <th>address</th> <th>math</th> <th>english</th>	age	sex	address	math	english
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
6	刘德华	57	男	香港	99	99

3.最后统计每组的人数

sex	COUNT(*)
女	1
男	3

- 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:

-- 对分组查询的结果再进行过滤

```
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

id	name	age	sex	address	math	english
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

1.先过滤掉年龄小于25岁的人。

id	name	age	sex	address	math	english
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
6	刘德华	57	男	香港	99	99

2.再分组。

id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
6	刘德华	57	男	香港	99	99

3.统计每组的人数

sex	COUNT (*)
女	1
男	3

4.显示性别人数大于2的数据

sex	COUNT (*)
男	3

3.3.1 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

3.3.2 面试题：有如下订单表

Orders 表数据如下所示，执行如下 SQL 语句，运行结果是？

id	product	price
1	纸巾	16
2	纸巾	16
3	红牛	5
4	洗衣粉	60
5	苹果	8
6	洗衣粉	60

```
select product,sum(price)
from orders group by product where sum(price) > 30;
```

运行有误，group by 后面不能出现 where，使用 having

3.4 limit 语句

3.4.1 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);
```

3.4.2 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];
```

3.4.3 LIMIT 语法格式：

```
LIMIT offset,length;
```

offset: 起始行数，从 0 开始计数，如果省略，默认就是 0

length: 返回的行数

-- 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;
```

所有数据						
id	name	age	sex	address	math	english
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
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

跳过2条记录 → 显示6条记录

SELECT * FROM student3 LIMIT 2,6;

id	name	age	sex	address	math	english
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

3.4.4 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。假设我们每页显示 5 条记录的方式来分页。

- SQL 语句如下：

所有数据

id	name	age	sex	address	math	english
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
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

SELECT * FROM student3 LIMIT 0,5;

id	name	age	sex	address	math	english
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)

SELECT * FROM student3 LIMIT 5,5;

id	name	age	sex	address	math	english
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
9	唐僧	25	男	长安	87	78
10	孙悟空	18	男	花果山	100	66

SELECT * FROM student3 LIMIT 10,5;

id	name	age	sex	address	math	english
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

-- 如果第一个参数是 0 可以省略写：

```
select * from student3 limit 5;
```

```
-- 最后如果不够 5 条，有多少显示多少
```

```
select * from student3 limit 10,5;
```

第4节 数据库备份和还原

4.1 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。

4.2 备份与还原的语句

4.2.1 备份格式：DOS下，未登录的时候。这是一个可执行文件exe，在bin文件夹

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

4.2.2 还原格式：mysql中的命令，需要登录后才可以操作

```
USE 数据库;
```

```
SOURCE 导入文件的路径;
```

4.2.3 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

4.2.4 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。

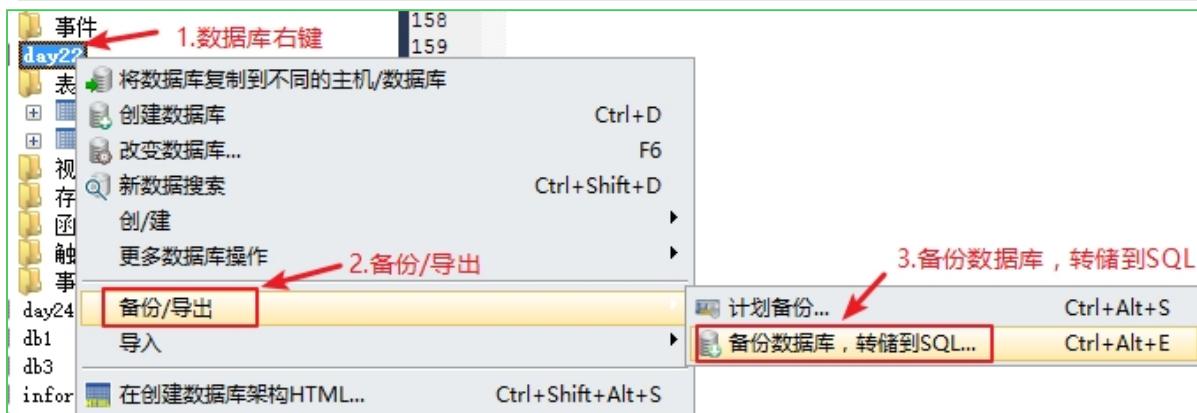
- 1) 删除 day21 数据库中的所有表
- 2) 登录 MySQL
- 3) 选中数据库
- 4) 使用 SOURCE 命令还原数据
- 5) 查看还原结果

```
use day21;
source d:/day21.sql;
```

4.3 图形化界面备份与还原

4.3.1 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。



4.3.2 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





第5节 数据库表的约束

5.1 数据库约束的概述

5.1.1 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。一个表如果添加了约束，不正确的数据将无法插入到表中。约束在创建表的时候添加比较合适。

5.1.2 约束种类：

约束名	约束关键字
主键	primary key
唯一	unique
非空	not null
外键	foreign key
检查约束	check 注：mysql 不支持

5.2 主键约束

5.2.1 主键的作用

用来唯一标识数据库中的每一条记录

NAME	age	score	id	NAME	age	score
张三	20	80	1	张三	20	80
李四	21	90	2	李四	21	90
王五	19	70	3	王五	19	70
张三	20	80	4	张三	20	80

5.2.2 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。**

如：身份证号，学号不建议做成主键

5.2.3 创建主键

- 主键关键字： primary key

- 主键的特点：

- 1) 非空 not null

2) 唯一

- 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

2. 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 st5, 包含字段(id, name, age)将 id 做为主键

```
create table st5 (
    id int primary key, -- id 为主键
    name varchar(20),
    age int
)
```

```
desc st5;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	7B	NO	PRI	(NULL) OK
name	varchar(20)	11B	YES		(NULL) OK
age	int(11)	7B	YES		(NULL) OK

-- 插入重复的主键值

```
insert into st5 values (1, '关羽', 30);
-- 错误代码: 1062 Duplicate entry '1' for key 'PRIMARY'
insert into st5 values (1, '关云长', 20);

select * from st5;
```

```
-- 插入 NULL 的主键值, Column 'id' cannot be null
insert into st5 values (null, '关云长', 20);
```

5.2.4 删除主键

-- 删除 st5 表的主键

```
alter table st5 drop primary key;
```

-- 添加主键

```
alter table st5 add primary key(id);
```

5.2.5 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

AUTO_INCREMENT 表示自动增长(字段类型必须是整数类型)

-- 插入数据

```
insert into st6 (name,age) values ('小乔',18);
insert into st6 (name,age) values ('大乔',20);
-- 另一种写法
insert into st6 values(null,'周瑜',35);
```



```
select * from st6;
```

主键开始从1自动增长的

id	NAME	age
1	唐僧	22
2	孙悟空	26
3	猪八戒	25
4	沙僧	20

5.2.6 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;
```

```
-- 指定起始值为 1000  
create table st4 (  
    id int primary key auto_increment,  
    name varchar(20)  
) auto_increment = 1000;  
  
insert into st4 values (null, '孔明');  
select * from st4;
```

- 创建好以后修改起始值

```
ALTER TABLE 表名 AUTO_INCREMENT=起始值;
```

```
alter table st4 auto_increment = 2000;  
insert into st4 values (null, '刘备');
```

	id	name
	1000	孔明
	2000	刘备

5.2.7 DELETE 和 TRUNCATE 对自增长的影响

- DELETE: 删除所有的记录之后, 自增长没有影响。

	id	name	age
	4	小乔	18
	5	大乔	20
	6	周瑜	35

- TRUNCATE: 删除以后, 自增长又重新开始。

	id	name	age
	1	小乔	18
	2	大乔	20
	3	周瑜	35

5.3 唯一约束

什么是唯一约束： 表中某一列不能出现重复的值

5.3.1 唯一约束的基本格式

字段名 字段类型 UNIQUE

5.3.2 实现唯一约束

-- 创建学生表 st7, 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 st7 (
    id int,
    name varchar(20) unique
)
```

-- 添加一个同名的学生

```
insert into st7 values (1, '张三');

select * from st7;
-- Duplicate entry '张三' for key 'name'
insert into st7 values (2, '张三');
```

-- 重复插入多个 null 会怎样?

```
insert into st7 values (2, null);

insert into st7 values (3, null);
```

id	name
1	张三
2	(NULL)
3	(NULL)

null 没有数据, 不存在重复的问题

5.4 非空约束

- 什么是非空约束： 某一列不能为 null。

5.4.1 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 st8, 包含字段(id, name, gender) 其中 name 不能为 NULL

```
create table st8 (
    id int,
    name varchar(20) not null,
    gender char(1)
)
```

-- 添加一条记录其中姓名不赋值

```
insert into st8 values (1, '张三疯', '男');

select * from st8;
```

```
-- Column 'name' cannot be null
insert into st8 values (2,null,'男');
```

5.4.2 默认值

什么是默认值：

字段名 字段类型 DEFAULT 默认值

```
-- 创建一个学生表 st9, 包含字段(id, name, address), 地址默认值是广州
```

```
create table st9 (
    id int,
    name varchar(20),
    address varchar(20) default '广州'
)
```

```
-- 添加一条记录, 使用默认地址
```

```
insert into st9 values (1, '李四', default);
select * from st9;
```

```
insert into st9 (id, name) values (2, '李白');
```

```
-- 添加一条记录, 不使用默认地址
```

```
insert into st9 values (3, '李四光', '深圳');
```

- 疑问：如果一个字段设置了非空与唯一约束，该字段与主键的区别？

- 1) 主键数在一个表中，只能有一个。不能出现多个主键。主键可以单列，也可以是多列。
- 2) 自增长只能用在主键上

5.5 外键约束

5.5.1 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 5 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');
```

- 以上数据表的缺点：

- 1) 数据冗余

2) 后期还会出现增删改的问题

id	name	age	dep_name	dep_location
1	张三	20	研发部	广州
2	李四	21	研发部	广州
3	王五	20	研发部	广州
4	老王	20	销售部	深圳
5	大王	22	销售部	深圳
6	小王	18	销售部	深圳

5.5.2 解决方案：

员工通过dep_id去部门表中找到对应的部门

employee员工表

id	NAME	age	dep_id
1	张三	20	1
2	李四	21	1
3	王五	20	1
4	老王	20	2
5	大王	22	2
6	小王	18	2

department部门表

id	dep_name	dep_location
1	研发部	广州
2	销售部	深圳

-- 解决方案：分成 2 张表

-- 创建部门表(id,dep_name,dep_location)

-- 一方，主表

```
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)

-- 多方，从表

```
create table employee(
    id int primary key auto_increment,
    name varchar(20),
    age int,
    dep_id int -- 外键对应主表的主键
)
```

-- 添加 2 个部门

```
insert into department values(null, '研发部', '广州'), (null, '销售部', '深圳');
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 ('小王', 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

id	NAME	age	dep_id
1	张三	20	1
2	李四	21	1
3	王五	20	1
4	老王	20	2
5	大王	22	2
6	小王	18	2
7	老张	18	6

id	dep_name	dep_location
1	研发部	广州
2	销售部	深圳

这条记录的
dep_id有问题

- 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

5.5.3 什么是外键约束

- 什么是外键：在从表中与主表主键对应的那一列，如：员工表中的 `dep_id`
- 主表：一方，用来约束别人的表
- 从表：多方，被别人约束的表

一张表中的某个字段引用另一个表的主键

employee 员工表

id	NAME	age	dep_id
1	张三	20	1
2	李四	21	1
3	王五	20	1
4	老王	20	2
5	大王	22	2
6	小王	18	2

外键

主键

department 部门表

id	dep_name	dep_location
1	研发部	广州
2	销售部	深圳

主表：约束别人

副表/从表：使用别人的数据，被别人约束

5.5.4 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);

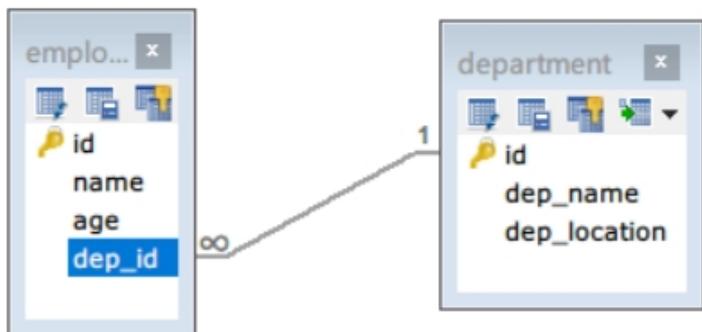
- 具体操作：

```
-- 1) 删除副表/从表 employee
drop table employee;
```

```
-- 2) 创建从表 employee 并添加外键约束 emp_dep_id_fk
-- 多方, 从表
create table employee(
    id int primary key auto_increment,
    name varchar(20),
    age int,
    dep_id int, -- 外键对应主表的主键
    -- 创建外键约束
    constraint emp_dep_id_fk foreign key (dep_id) references department(id)
)
-- 3) 正常添加数据
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);

select * from employee;
-- 4) 部门错误的数据添加失败
-- 插入不存在的部门
-- Cannot add or update a child row: a foreign key constraint fails
INSERT INTO employee (NAME, age, dep_id) VALUES ('老张', 18, 6);
```



5.5.5 删删除外键

ALTER TABLE 从表 drop foreign key 外键名称;

```
-- 删除 employee 表的 emp_dep_id_fk 外键
alter table employee drop foreign key emp_dep_id_fk;

-- 在 employee 表存在的情况下添加外键
alter table employee add constraint emp_dep_id_fk
foreign key (dep_id) references department(id);
```

5.5.6 外键的级联

- 出现新的问题:

```
select * from employee;
```

```
select * from department;  
-- 要把部门表中的 id 值 2, 改成 5, 能不能直接更新呢?  
-- Cannot delete or update a parent row: a foreign key constraint fails  
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: a foreign key constraint fails  
delete from department where id=1;
```

● 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，只能是创建表的时候创建级联关系。更新主表中的主键，从表中的外键列也自动同步更新
ON DELETE CASCADE	级联删除

```
-- 删除 employee 表，重新创建 employee 表，添加级联更新和级联删除
```

```
drop table employee;
```

```
create table employee(  
id int primary key auto_increment,  
name varchar(20),  
age int,  
dep_id int, -- 外键对应主表的主键  
-- 创建外键约束  
constraint emp_dep_id_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);
```

```
-- 删除部门表？能不能直接删除？
```

```
drop table department;
```

```
-- 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;  
select * from employee;  
select * from department;
```

```
-- 删除部门号是 2 的部门
```

```
delete from department where id=2;
```

5.6 数据约束小结

约束名	关键字	说明
主键	primary key	1) 唯一 2) 非空
默认	default	如果一列没有值，使用默认值
非空	not null	这一列必须有值
唯一	unique	这一列不能有重复值
外键	foreign key	主表中主键列，在从表中外键列

第6节 表与表之间的关系

6.1 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
一对多：最常用的关系 部门和员工
多对多：学生选课表 和 学生表， 一门课程可以有多个学生选择，一个学生选择多门课程
一对一：相对使用比较少。员工表 简历表， 公民表 护照表

6.2 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



6.3 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。

张三选择语文和数学

李四选择数学和英语

多对多

学生表

学号	姓名
1	张三
2	李四
3	王五

中间表

学生-课程关系表

学号	课程号
1	1
1	2
2	2
2	3
3	3

课程表

课程号	课程名
1	语文
2	数学
3	英语

6.4 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。

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	主表的主键和从表的主键，形成主外键关系

一对一

学生表

学号	姓名	简历号
1	张三	1
2	李四	2
3	王五	3

简历表

简历号	简历
1	张三的简历
2	李四的简历
3	王五的简历

一对一

学生表

学号	姓名
1	张三
2	李四
3	王五

个人信息

编号	出生地	曾用名	出生体重
1	广东	四毛	6.2
2	广西	三毛	5.5
3	江西	小王	7.3

6.5 一对多关系案例

6.5.1 需求：一个旅游线路分类中有多个旅游线路

- 界面

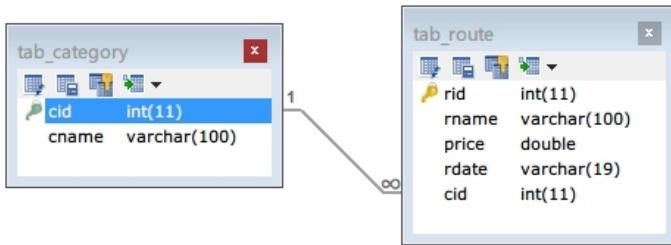


The screenshot shows a travel tour listing. At the top, there's a navigation bar with links like 首页, 特卖汇, 门票, 酒店, 香港车票, 周边游, 出境游, 国内游, 港澳游, 包团定制, 全球自由行. The '港澳游' link is highlighted with a red arrow. Below the navigation, a breadcrumb trail says '首页 > 国内游 > [2月 桂林龙脊梯田放水节+阳朔高铁3天·全程0自费] 龙脊梯田+平安寨梯田+阳朔葡萄峰林+遇龙河风光【豪叹杀猪宴】'. The main content area features a large image of terraced fields, a detailed description of the tour, contact information for the operator (Jinma Travel), and a price of 739. There are also buttons for '收藏' (Collection) and '点击收藏' (Click to Collection).

- 表与表的关系

rid	rname	price	rdate	cid
1	【厦门+鼓浪屿+南普陀寺+曾厝垵 高铁3天 惠贵团】	1499	2018-01-27	1
2	【浪漫桂林 阳朔西街高铁3天纯玩 高级团】	699	2018-02-22	3
3	【爆款 ¥ 1699秒杀】泰国 曼谷 芭提雅 金沙岛 杜拉拉岛	1699	2018-01-27	2
4	【经典•狮航 ¥ 2399秒杀】巴厘岛双飞五天 抵玩【广州	2399	2017-12-23	2
5	香港迪士尼乐园自由行2天【永东跨境巴士广东至迪士尼	799	2018-04-10	4

cid	cname
1	周边游
2	出境游
3	国内游
4	港澳游



6.5.2 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:
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
*/
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 热卖】泰国 曼谷 芭提雅 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;
```

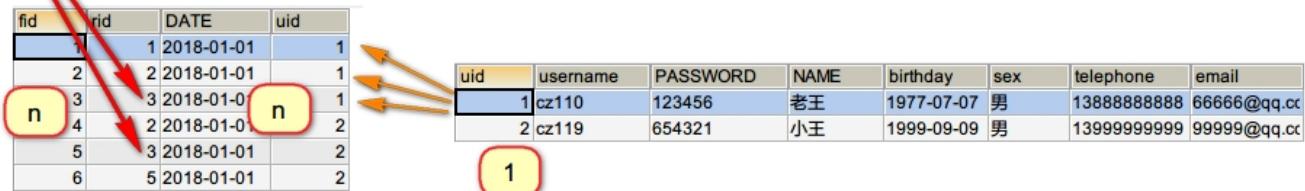
6.6 多对多关系案例

6.6.1 需求：一个用户收藏多个线路，一个线路被多个用户收藏

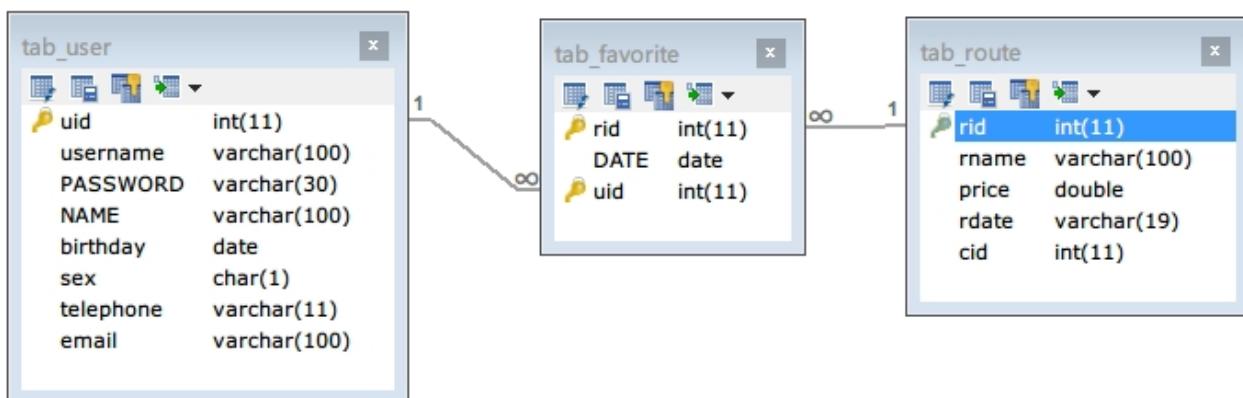
用户中心 > 我的收藏



rid	rname	price	rdate	cid
1	【厦门+鼓浪屿+南普陀寺+曾厝垵 高铁3天 惠贵团】尝	1499	2018-01-27	1
2	【浪漫桂林 阳朔西街高铁3天纯玩 高级团】城徽象鼻山	699	2018-02-22	3
3	【爆款￥1699秒杀】泰国 曼谷 芭提雅 金沙岛 杜拉拉水	1699	2018-01-27	2
4	【经典•狮航 ￥2399秒杀】巴厘岛双飞五天 抵玩【广州	2399	2017-12-23	2
5	香港迪士尼乐园自由行2天【永东跨境巴士广东至迪士尼	799	2018-04-10	4



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的关系



6.6.2 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
*/
create table tab_favorite (
    rid int,
    date datetime,
    uid int,
    -- 创建复合主键
    primary key(rid,uid),
    foreign key (rid) references tab_route(rid),
    foreign key(uid) references tab_user(uid)
)

-- 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
```

```
select * from tab_favorite;
```

6.7 表与表之间的关系小结

表与表的关系 关系的维护	
一对多	主外键的关系
多对多	中间表，两个一对多
一对一	1) 特殊一对多，从表中的外键设置为唯一 2) 从表中的主键又是外键

第7节 数据库设计

7.1 数据规范化

7.1.1 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。

7.1.2 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。一般说来，数据库只需满足第三范式（3NF）就行了。

7.2 1NF

7.2.1 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。**

7.2.2 班级表

学号	姓名	班级
1	张三	一年三班
2	李四	一年二级
3	王五	二年三班

7.3 2NF

7.3.1 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。

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主键列的时候，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：

- 1) 一张表只描述一件事情。
- 2) 表中的每一列都完全依赖于主键

7.3.2 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) 分成两张表

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

7.4 3NF：

7.4.1 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 " $A \rightarrow B \rightarrow C$ " 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 \rightarrow 非主键列 $x \rightarrow$ 非主键列 y

7.4.2 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 \rightarrow 所在学院 \rightarrow 学院地点

- 拆分成两张表

学号	姓名	年龄	所在学院的编号(外键)
----	----	----	-------------

学院编号	所在学院	学院地点
------	------	------

7.4.3 三大范式小结：

范式	特点
1NF	原子性：表中每列不可再拆分。
2NF	不产生局部依赖，一张表只描述一件事情
3NF	不产生传递依赖，表中每一列都直接依赖于主键。而不是通过其它列间接依赖于主键。