MySQL（六）

1. 子查询

盒子里面放盒子，里面的盒子可以叫子盒子。

1.1 什么是子查询

子查询，在一条 SQL 语句中嵌入另一条 SQL 语句。

- 主查询
  我们管最外层的查询叫做主查询。
- 子查询
  嵌入的 SQL 叫做子查询。

1.2 子查询分类

1.2.1 按返回结果分

标量子查询：结果返回一个值。（一行一列）

列子查询：返回结果是一列。（多行一列）
行子查询：返回结果是一行。（一行多列）
表子查询：返回结果是多行多列。
exists 子查询：返回的结果是 1 或者 0。（类似布尔操作）

1.2.2 按位置分

where 子查询：子查询出现在 where 的位置上，用子查询的结果做为条件。
from 子查询：子查询出现在 from 的位置上，用子查询的结果做为一张表。
select 子查询：出现在 select 后面列的位置上，用子查询的结果做为一列。

1.3 准备数据

```
mysql> select * from class;
+----+----------------+
| id | name           |
+----+----------------+
|  1 | 全栈开发班     |
|  2 | Java开发班     |
|  3 | 服装设计班     |
|  4 | 美容美发班     |
|  5 | 挖掘机精英班   |
|  6 | 美国总统速成班 |
+----+----------------+
6 rows in set (0.00 sec)

mysql> select * from students;
+----+---------+--------+-------------+----------+-----+--------+
| id | name    | gender | tel         | class_id | age | height |
+----+---------+--------+-------------+----------+-----+--------+
|  1 | 八戒    | 男     | 13912345555 |        1 |  20 |    180 |
|  2 | 大师兄  | 男     | 13912346666 |        1 |  21 |    175 |
|  3 | 金角大王 | 男     | 13922222222 |        2 | 204 |    195 |
|  4 | 西施    | 女     | 13912342343 |        4 |  17 |    165 |
|  5 | 大乔    | 女     | 13912346666 |        3 |  22 |    168 |
|  6 | 貂蝉    | 女     | 13922233222 |        1 |  21 |    164 |
|  7 | 大乔    | 男     | 13922225454 |        2 |  19 |    183 |
+----+---------+--------+-------------+----------+-----+--------+
7 rows in set (0.00 sec)
```

```
create database subquery default character set utf8;
use subquery;

create table class
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '班级',
    primary key(id)
) comment='班级表';

    insert into class
        values
        (1,'全栈开发班'),
        (2,'Java 开发班'),
        (3,'服装设计班'),
        (4,'美容美发班'),
        (5,'挖掘机精英班'),
        (6,'美国总统速成班');

# 学生表
create table students
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '姓名',
    gender enum('男','女') not null comment '性别',
    tel bigint unsigned not null comment '手机号',
    class_id int unsigned not null comment '班级 Id',
    age tinyint unsigned not null comment '年龄',
   height tinyint unsigned not null comment '身高，单位：厘米',
    primary key(id)
) comment='学生表';

    insert into students
    values
    (1,'八戒','男',13912345555,1,20,180),
    (2,'大师兄','男',13912346666,1,21,175),
    (3,'金角大王','男',13922222222,2,204,195),
    (4,'西施','女',13912342343,4,17,165),
    (5,'大乔','女',13912346666,3,22,168),
    (6,'貂蝉','女',13922233222,1,21,164),
    (7,'大乔','男',13922225454,2,19,183);
```

2. 标量子查询

标量子查询，子查询返回的结果是一个值【一行一列】

- **示例：取出“八戒”所在的班级的名称。**

```
mysql> select name from class where id = (select class_id from students where name = '八戒');
+------------+
| name       |
+------------+
| 全栈开发班 |
+------------+
1 row in set (0.00 sec)
```

说明：子查询返回的是一个值：【一行一列】

```
mysql> select class_id from students where name = '八戒';
+----------+
| class_id |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

所以我们叫它标量子查询。

3. 列子查询

列子查询，子查询返回结果是一列。

注意：如果子查询返回的结果是一列的话，那么需要使用 in 连接不能是= 。

- **示例：取出所有女同学所在的班级名称。**

第一步：取出女同学所在的班级的 ID【结果是一列】

```
mysql> select class_id from students where gender = '女';
+----------+
| class_id |
+----------+
|        4 |
|        3 |
|        1 |
+----------+
3 rows in set (0.00 sec)
```

第二步：从班级表中取出班级名称

```
mysql> select name from class where id in (select class_id from students where gender = '女');
+------------+
| name       |
+------------+
| 美容美发班 |
| 服装设计班 |
| 全栈开发班 |
+------------+
3 rows in set (0.00 sec)
```

因为子查询的结果是一列，所以需要使用 in 而不是=

4. 行子查询

行子查询，子查询返回的是一行的数据。

- **示例：找出年龄最大并且身高也是最高的同学。**

```
-- 找出这样一个同学：年龄是最大，并且身高也是最高
    # 第一步：取出班级最大的年龄 和 最大的身高
        select max(age),max(height) from students;
        +----------+-------------+
        | max(age) | max(height) |
        +----------+-------------+
        |      204 |         195 |
        +----------+-------------+
        1 row in set (0.00 sec)
    # 第二步：取出年龄=204，并且身高=195
        select * from students where (age,height) = (select max(age),max(height) from students);
```

返回的结果是一行

5. 表子查询

表子查询，子查询返回的结果是多行多列的数据。

- **示例：取出每个班级中年龄最大的同学。**

思路：先通过子查询把数据根据年龄子降序排列，然后再套一层 SQL 对其进行分组。【因为一条 SQL 中是先分组后排序，无法实现先排序后分组】

```
# 第一步：把同一个班级的同学放到时一起【根据class_分组】
    select * from students group by class_id

        1. 先分组
        +----+----------+--------+-------------+----------+-----+--------+
        | id | name     | gender | tel         | class_id | age | height |
        +----+----------+--------+-------------+----------+-----+--------+
        |  1 | 八戒     | 男     | 13912345555 |        1 |  20 |    180 |
        |  2 | 大师兄   | 男     | 13912346666 |        1 |  21 |    175 |
        |  6 | 貂蝉     | 女     | 13922233222 |        1 |  21 |    164 |

        |  3 | 金角大王 | 男     | 13922222222 |        2 | 204 |    195 |
        |  7 | 大乔     | 男     | 13922225454 |        2 |  19 |    183 |

        |  4 | 西施     | 女     | 13912342343 |        4 |  17 |    165 |

        |  5 | 大乔     | 女     | 13912346666 |        3 |  22 |    168 |
        +----+----------+--------+-------------+----------+-----+--------+
        2. mysql只保留每一组中的第一条记录
        +----+----------+--------+-------------+----------+-----+--------+
        | id | name     | gender | tel         | class_id | age | height |
        +----+----------+--------+-------------+----------+-----+--------+
        |  1 | 八戒     | 男     | 13912345555 |        1 |  20 |    180 |
        |  3 | 金角大王 | 男     | 13922222222 |        2 | 204 |    195 |
        |  4 | 西施     | 女     | 13912342343 |        4 |  17 |    165 |
        |  5 | 大乔     | 女     | 13912346666 |        3 |  22 |    168 |
        +----+----------+--------+-------------+----------+-----+--------+
```

```
        3. 因为分组只保留每一组中的第一条记录，而第一条记录并不是年龄最大，所以如果我们先根据age字段
        子段降序排列，然后再分组，这时候每一组中的第一条记录肯定是这一组中年龄最大的。
        所以我们应该 先排序再分组，但是，因为在一条SQL语句中是先分组然后才排序的，而我们要先排序后分
        组，所以一条SQL不行，我们需要写两条。
        select * from students order by age desc
        +----+----------+--------+-------------+----------+-----+--------+
        | id | name     | gender | tel         | class_id | age | height |
        +----+----------+--------+-------------+----------+-----+--------+
        |  3 | 金角大王 | 男     | 13922222222 |        2 | 204 |    195 |
        |  5 | 大乔     | 女     | 13912346666 |        3 |  22 |    168 |
        |  2 | 大师兄   | 男     | 13912346666 |        1 |  21 |    175 |
        |  6 | 貂蝉     | 女     | 13922233222 |        1 |  21 |    164 |
        |  1 | 八戒     | 男     | 13912345555 |        1 |  20 |    180 |
        |  7 | 大乔     | 男     | 13922225454 |        2 |  19 |    183 |
        |  4 | 西施     | 女     | 13912342343 |        4 |  17 |    165 |
        +----+----------+--------+-------------+----------+-----+--------+
        7 rows in set (0.00 sec)
```

```
4. 我们把上面这个整个结果做为一个子查询，再其基础上，进行分组，
    注意：当子查询做为数据来源时，需要起个别名，比如 c
select * from (select * from students order by age desc) as c group by class_id
+----+----------+--------+-------------+----------+-----+--------+
| id | name     | gender | tel         | class_id | age | height |
+----+----------+--------+-------------+----------+-----+--------+
|  1 | 八戒     | 男     | 13912345555 |        1 |  20 |    180 |
|  3 | 金角大王 | 男     | 13922222222 |        2 | 204 |    195 |
|  5 | 大乔     | 女     | 13912346666 |        3 |  22 |    168 |
|  4 | 西施     | 女     | 13912342343 |        4 |  17 |    165 |
+----+----------+--------+-------------+----------+-----+--------+
```

6. exists 子查询

子查询是否返回结果，返回结果为真，没有结果为假。

- **示例：取出有人报名的班级名称。**

```
--取出有人报名的班级名称
    思路：用班级的id到学生表中去找，如果有学生的class_id和这个班级的ID相同，说明有人报名。
    # 每个班级都到 子查询中找有没有 对应的学生，如果有就显示，否则就过滤掉
    select * from class a where exists (select id from students b where a.id=b.class_id);
    +----+------------+
    | id | name       |
    +----+------------+
    |  1 | 全栈开发班 |
    |  2 | Java开发班 |
    |  3 | 服装设计班 |
    |  4 | 美容美发班 |
    +----+------------+
```

7. 表引擎

7.1 查看 MySQL 中可用的表引擎

可以使用 show engines 指令查看 MySQL 服务器支持的表引擎：

```
mysql> show engines;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| InnoDB             | YES     | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| MyISAM             | DEFAULT | MyISAM storage engine                                          | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
9 rows in set (0.00 sec)
```

7.2 MyISAM 引擎

Storage limits	256TB	*Transactions*	No	*Locking granularity*	Table
MVCC	No	*Geospatial data type support*	Yes	*Geospatial indexing support*	Yes
B-tree indexes	Yes	*T-tree indexes*	No	*Hash indexes*	No
Full-text search indexes	Yes	*Clustered indexes*	No	*Data caches*	No
Index caches	Yes	*Compressed data*	Yes [a]	*Encrypted data* [b]	Yes
Cluster database support	No	*Replication support* [c]	Yes	*Foreign key support*	No
Backup / point-in-time recovery [d]	Yes	*Query cache support*	Yes	*Update statistics for data dictionary*	Yes

存储限制：256TB。【1TB=1024GB】
事务：不支持
锁的粒度：表级
外键：不支持

7.3 InnoDB 引擎

注意说明：MySQL 官方网站推荐我们使用这种引擎。

Storage limits	64TB	*Transactions*	Yes	*Locking granularity*	Row
MVCC	Yes	*Geospatial data type support*	Yes	*Geospatial indexing support*	Yes [a]
B-tree indexes	Yes	*T-tree indexes*	No	*Hash indexes*	No [b]
Full-text search indexes	Yes [c]	*Clustered indexes*	Yes	*Data caches*	Yes
Index caches	Yes	*Compressed data*	Yes [d]	*Encrypted data* [e]	Yes
Cluster database support	No	*Replication support* [f]	Yes	*Foreign key support*	Yes
Backup / point-in-time recovery [g]	Yes	*Query cache support*	Yes	*Update statistics for data dictionary*	Yes

存储限制：64TB。【1TB=1024GB】
事务：支持
锁的粒度：行级
外键：支持

7.4 设置表引擎

在 create table 创建表时可以通过 engine 属性设置表引擎：

```
create table 表名 (
   字段...
)type|engine=引擎名;
```

8. 事务

8.1 什么是事务？

事务，就是“要完成的一件事情”，比如购物、转账、做饭、洗澡等等。
做一件事情通常由多个步骤组成，如果某个步骤失败了，那就尴尬了.....

有时我们要完成一个功能，需要执行多个 SQL 语句，如果这些 SQL 执行到一半突然停电了，那么就会导致这个功能只完成了一半，这种情况在很多时候是不允许出现的，比如：银行转账时。

MySQL 中提供了事务的功能，可以确保一个事务中的多条 SQL 语句要就都成功，要就都失败，不会在中途失败。

注意：MySQL 中只有 InnoDB 引擎的表才支持事务功能。

8.2 转账功能

8.2.1 创建表结构

```
# 数据库
create database shiwu default character set utf8;
use shiwu;
# 账号表
create table users
(
    id mediumint unsigned not null auto_increment comment '编号',
    username varchar(50) not null comment '用户名',
    password char(32) not null comment '密码',
    money decimal(10,2) not null default '0.00' comment '钱',
    primary key (id)
)engine=InnoDB;
# 插入测试数据
insert into users
values(1,'八戒','123456',0),
(2,'大师兄','123456',2000);
```

8.2.2 转账功能

“大师兄给八戒转账 100 元钱”。

实现这个功能需要以下两条 SQL 语句：

```
update users set money=money-100 where id = 2; # 大师兄减 100 元钱
update users set money=money+100 where id = 1; # 八戒加 100 元钱
```

如果第一条记录执行完之后突然停电了，第二条 SQL 没有执行，那么大师兄白白丢了 100 元钱。

8.3 事务的使用

可以使用三个指令使用事务：

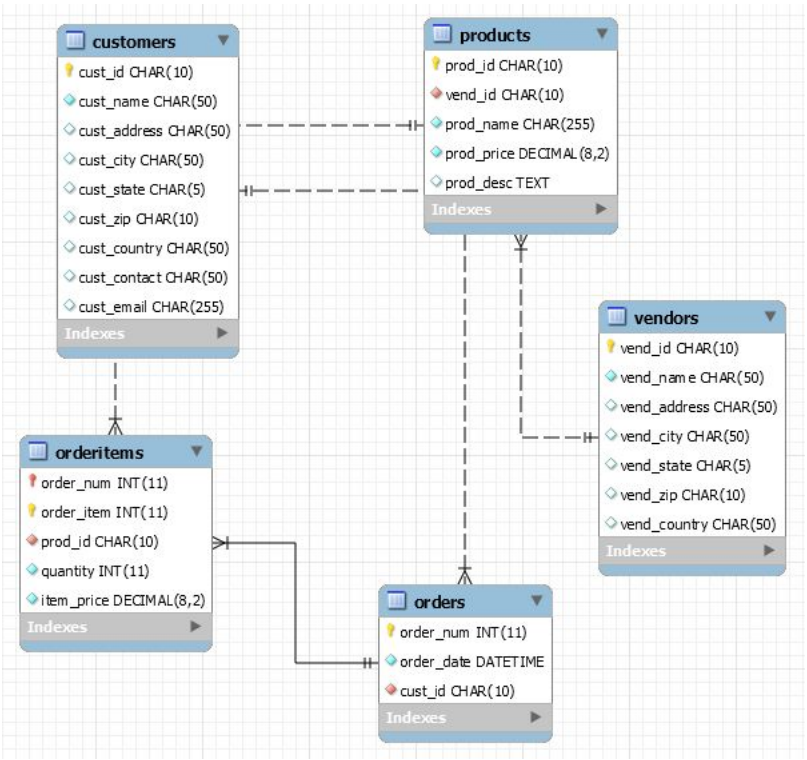
命令	描述
start transaction	开启一个事务。
commit	提交一个事务。
rollback	回滚【取消】一个事务。

```
start transaction;          # 开启一个事务
update users set money=money-100 where id = 2; # 大师兄减 100 元钱
update users set money=money+100 where id = 1; # 八戒加 100 元钱
commit;                     # 提交事务
```

实现原理：MySQL 在执行一个事务之前会先把 SQL 写到“事务日志”中，如果所有 SQL 都成功了才会把数据同步到数据表中，如果中途发生错误，MySQL 会根据事务日志回滚已经完成的操作。

9. 外键

外键，建立表间关系时必不可少的东西。



9.1 什么是外键

外键（foreign key），就是另一个表的主键，用来和另一个表建立关系。
注意：只有 InnoDB 引擎支持外键。

```
mysql> select * from class;
+----+----------------+
| id | name           |
+----+----------------+
|  1 | 全栈开发班     |
|  2 | Java开发班     |
|  3 | 服装设计班     |
|  4 | 美容美发班     |
|  5 | 挖掘机精英班   |
|  6 | 美国总统速成班 |
+----+----------------+
6 rows in set (0.00 sec)

mysql> select * from students;
+----+---------+--------+-------------+----------+-----+--------+
| id | name    | gender | tel         | class_id | age | height |
+----+---------+--------+-------------+----------+-----+--------+
|  1 | 八戒    | 男     | 13912345555 |        1 |  20 |    180 |
|  2 | 大师兄  | 男     | 13912346666 |        1 |  21 |    175 |
|  3 | 金角大王 | 男     | 13922222222 |        2 | 204 |    195 |
|  4 | 西施    | 女     | 13912342343 |        4 |  17 |    165 |
|  5 | 大乔    | 女     | 13912346666 |        3 |  22 |    168 |
|  6 | 貂蝉    | 女     | 13922233222 |        1 |  21 |    164 |
|  7 | 大乔    | 男     | 13922225454 |        2 |  19 |    183 |
+----+---------+--------+-------------+----------+-----+--------+
7 rows in set (0.00 sec)
```

外键：用来和另一个表建立关系的字段。

9.2 定义外键

- 添加外键

建添时加外键

```
    [constraint 名称] foreign key (字段名) references 表名(字段名)
```

constraint 为外键起一个名字。如果没有起名，MySQL 会默认给起个名字。

```
# 学生表
create table students
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '姓名',
    gender enum('男','女') not null comment '性别',
    tel bigint unsigned not null comment '手机号',
    class_id int unsigned not null comment '班级Id',
    age tinyint unsigned not null comment '年龄',
    primary key(id),
    forein key class_id references class(id)
) comment='学生表';
```

class_id字段是一个外键。
用来关系class表中的id字段。

为一个现有的表添加外键

```
    alter table 表名 add [constraint 名称] foreign key (字段名) references 表名(字
段名)
```

- 修改外键

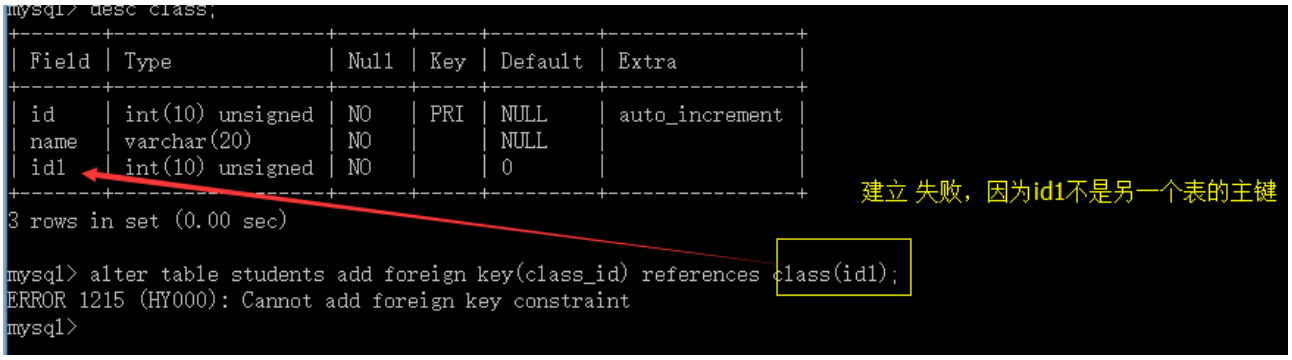
注意不能修改键，如果要修改我们只能先删除再重新添加。

- 删除外键

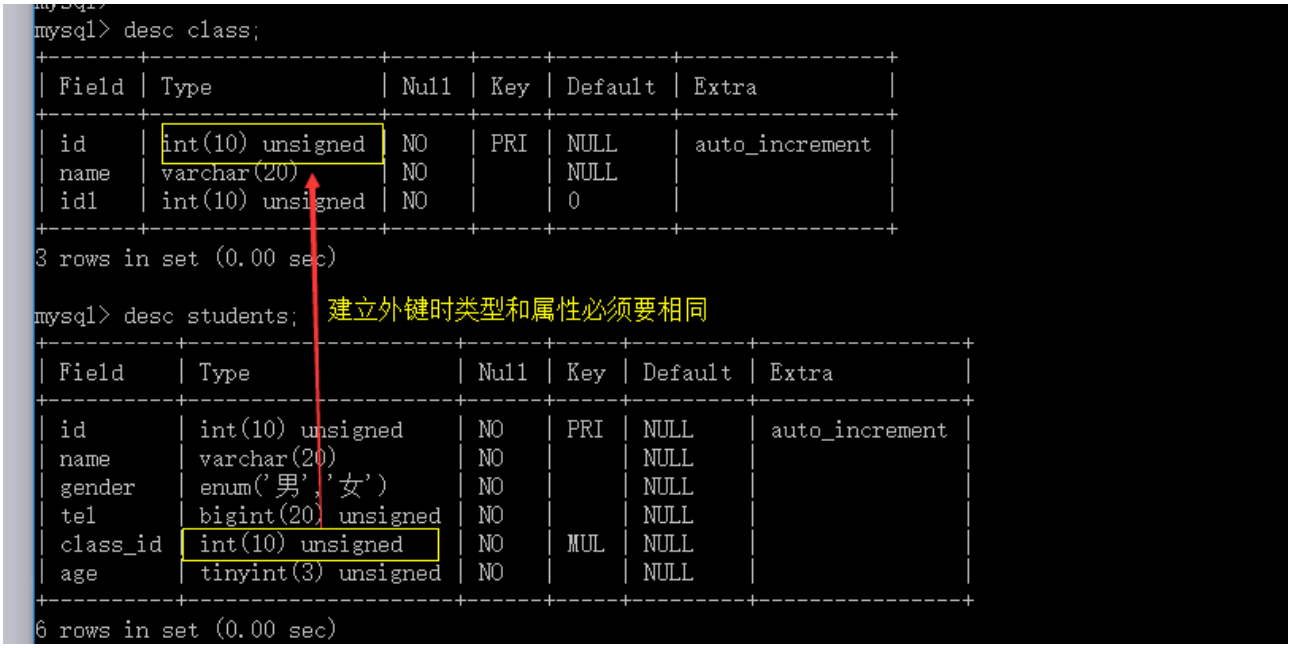
```
    alter table 表名 drop foreign key 外键名
```

9.3 外键的基本要求

1. 外键只能和另一个表中的主键建立关系



2. 建立关系的这两个字段必须字段的类型和属性都相同才可以。



9.4 外键约束

语法：

```
    foreign key (字段名) references 主表(主键) on 约束时机 约束模式
```

注意：约束的主键所在的表的操作。

约束时机：
update：修改时的约束。
delete：删除时的约束。

约束模式：
restrict：严格模式，不允许操作。
cascade：级别操作，从表和主表执行同样的操作。
set null：置空模式。
no action：什么也不做。

示例：当外键所在的表中有数据和键所在的表中的数据相关联，那么主键表中的数据是不允许被删除。

```
ysql>
ysql> alter table students add foreign key (class_id) references class(id) on delete restrict;
uery OK, 0 rows affected (0.08 sec)
ecords: 0  Duplicates: 0  Warnings: 0
```

10. 今日总结