16.1 准备数据

创建数据表

```
-- 创建 "京东" 数据库
create database jing_dong charset=utf8;

-- 使用 "京东" 数据库
use jing_dong;

-- 创建一个商品 goods 数据表
create table goods(
    id int unsigned primary key auto_increment not null,
    name varchar(150) not null,
    cate_name varchar(40) not null,
    brand_name varchar(40) not null,
    price decimal(10,3) not null default 0,
    is_show bit not null default 1,
    is_saleoff bit not null default 0
);
```

插入数据

```
-- 向 goods 表中插入数据

insert into goods values(0,'r510vc 15.6 英寸笔记本','笔记本','华硕','3399
',default,default);
insert into goods values(0,'y400n 14.0 英寸笔记本电脑','笔记本','联想','49
99',default,default);
insert into goods values(0,'g150th 15.6 英寸游戏本','游戏本','雷神','8499
',default,default);
insert into goods values(0,'x550cc 15.6 英寸笔记本','笔记本','华硕','2799
',default,default);
insert into goods values(0,'x240 超极本','超级本','联想','4880',default,
default);
insert into goods values(0,'u330p 13.3 英寸超极本','超级本','联想','4299',
default,default);
insert into goods values(0,'svp13226scb 触控超极本','超级本','索尼','7999
',default,default);
insert into goods values(0,'ipad mini 7.9 英寸平板电脑','平板电脑','苹果',
'1998',default,default);
insert into goods values(0,'ipad air 9.7 英寸平板电脑','平板电脑','苹果','
3388',default,default);
insert into goods values(0,'ipad mini 配备 retina 显示屏','平板电脑','苹
果','2788',default,default);
insert into goods values(0,'ideacentre c340 20 英寸一体电脑 ','台式机','
联想','3499',default,default);
```

```
insert into goods values(0,'vostro 3800-r1206 台式电脑','台式机','戴尔','
2899',default,default);
insert into goods values(0,'imac me086ch/a 21.5 英寸一体电脑','台式机','
苹果','9188',default,default);
insert into goods values(0,'at7-7414lp 台式电脑 linux ）','台式机','宏碁',
'3699',default,default);
insert into goods values(0,'z220sff f4f06pa 工作站','服务器/工作站','惠普
','4288',default,default);
insert into goods values(0,'poweredge ii 服务器','服务器/工作站','戴尔','
5388',default,default);
insert into goods values(0,'mac pro 专业级台式电脑','服务器/工作站','苹果',
'28888',default,default);
insert into goods values(0,'hmz-t3w 头戴显示设备','笔记本配件','索尼','69
99',default,default);
insert into goods values(0,'商务双肩背包','笔记本配件','索尼','99',defaul
t,default);
insert into goods values(0,'x3250 m4 机架式服务器','服务器/工作站','ibm','
6888',default,default);
insert into goods values(0,'商务双肩背包','笔记本配件','索尼','99',defaul
t,default);
```

16.2 SQL 演练

1. SQL 语句的强化

- 查询类型 cate_name 为'超极本'的商品名称、价格

```
select name,price from goods where cate_name = '超级本';
```

- 显示商品的种类

```
select cate_name from goods group by cate_name;
```

- 求所有电脑产品的平均价格,并且保留两位小数

```
select round(avg(price),2) as avg_price from goods;
```

- 显示每种商品的平均价格

```
select cate_name,avg(price) from goods group by cate_name;
```

- 查询每种类型的商品中 最贵、最便宜、平均价、数量

```
select cate_name,max(price),min(price),avg(price),count(*) from goods g
roup by cate_name;
```

- 查询所有价格大于平均价格的商品，并且按价格降序排序

```
select id,name,price from goods
where price > (select round(avg(price),2) as avg_price from goods)
order by price desc;
```

- 查询每种类型中最贵的电脑信息

```
select * from goods
inner join
    (
        select
        cate_name,
        max(price) as max_price,
        min(price) as min_price,
        avg(price) as avg_price,
        count(*) from goods group by cate_name
    ) as goods_new_info
on goods.cate_name=goods_new_info.cate_name and goods.price=goods_new_i
nfo.max_price;
```

2. 创建"商品分类"" 表

```
-- 创建商品分类表
create table if not exists goods_cates(
    id int unsigned primary key auto_increment,
    name varchar(40) not null
);
```

- 查询 goods 表中商品的种类

```
select cate_name from goods group by cate_name;
```

- 将分组结果写入到 goods_cates 数据表

```
insert into goods_cates (name) select cate_name from goods group by cat
e_name;
```

3. 同步表数据

- 通过 goods_cates 数据表来更新 goods 表（**为什么要这么做？**）

```
update goods as g inner join goods_cates as c on g.cate_name=c.name set
 g.cate_name=c.id;
```

4. 创建 "商品品牌表" 表

- 通过 create...select 来创建数据表并且同时写入记录,一步到位

```
-- select brand_name from goods group by brand_name;

-- 在创建数据表的时候一起插入数据
-- 注意：需要对 brand_name 用 as 起别名，否则 name 字段就没有值
create table goods_brands (
    id int unsigned primary key auto_increment,
    name varchar(40) not null) select brand_name as name from goods gro
up by brand_name;
```

5. 同步数据

- 通过 goods_brands 数据表来更新 goods 数据表（为什么要这么做？）

```
update goods as g inner join goods_brands as b on g.brand_name=b.name s
et g.brand_name=b.id;
```

6. 修改表结构

- 查看 goods 的数据表结构,会发现 cate_name 和 brand_name 对应的类型为 varchar 但是存储的都是数字

```
desc goods;
```

- 通过 alter table 语句修改表结构

```
alter table goods
change cate_name cate_id int unsigned not null,
change brand_name brand_id int unsigned not null;
```

7. 外键

- 分别在 goods_cates 和 goods_brands 表中插入记录

```
insert into goods_cates(name) values ('路由器'),('交换机'),('网卡');
insert into goods_brands(name) values ('海尔'),('清华同方'),('神舟');
```

- 在 goods 数据表中写入任意记录

```
insert into goods (name,cate_id,brand_id,price)
values('LaserJet Pro P1606dn 黑白激光打印机', 12, 4,'1849');
```

- 查询所有商品的详细信息 (通过内连接)

```
select g.id,g.name,c.name,b.name,g.price from goods as g
inner join goods_cates as c on g.cate_id=c.id
inner join goods_brands as b on g.brand_id=b.id;
```

- 查询所有商品的详细信息 (通过左连接)

```
select g.id,g.name,c.name,b.name,g.price from goods as g
left join goods_cates as c on g.cate_id=c.id
left join goods_brands as b on g.brand_id=b.id;
```

- 如何防止无效信息的插入,就是可以在插入前判断类型或者品牌名称是否存在呢? **可以使用之前讲过的外键来解决**
- 外键约束:对数据的有效性进行验证
- 关键字: foreign key,**只有 innodb 数据库引擎 支持外键约束**
- 对于已经存在的数据表 如何更新外键约束

```
-- 给 brand_id 添加外键约束成功
alter table goods add foreign key (brand_id) references goods_brands(i
d);
-- 给 cate_id 添加外键失败
-- 会出现 1452 错误
-- 错误原因:已经添加了一个不存在的 cate_id 值 12,因此需要先删除
alter table goods add foreign key (cate_id) references goods_cates(id);
```

- 如何在创建数据表的时候就设置外键约束呢?
- 注意: goods 中的 cate_id 的类型一定要和 goods_cates 表中的 id 类型一致

```
create table goods(
    id int primary key auto_increment not null,
    name varchar(40) default '',
    price decimal(5,2),
    cate_id int unsigned,
    brand_id int unsigned,
    is_show bit default 1,
    is_saleoff bit default 0,
    foreign key(cate_id) references goods_cates(id),
    foreign key(brand_id) references goods_brands(id)
);
```

- 如何取消外键约束

```
-- 需要先获取外键约束名称,该名称系统会自动生成,可以通过查看表创建语句来获取名
称
```

```
show create table goods;
-- 获取名称之后就可以根据名称来删除外键约束
alter table goods drop foreign key 外键名称;
```

- **在实际开发中,很少会使用到外键约束,会极大的降低表更新的效率**

外键是否采用看业务应用场景，以及开发成本的，大致列下什么时候适合，什么时候不适合使用：

1. 互联网行业应用不推荐使用外键： 用户量大，并发度高，为此数据库服务器很容易成为性能瓶颈，尤其受 IO 能力限制，且不能轻易地水平扩展；**若是把数据一致性的控制放到事务(业务层)中**，也即让应用服务器承担此部分的压力，而应用（web 应用）服务器一般都是可以做到轻松地水平的伸缩；

2.传统行业
1>.软件应用的人数有限，换句话说是可控的；
2>.数据库服务器的数据量也一般不会超大，且活跃数据有限；

综合上述 2 句话描述，也即数据库服务器的性能不是问题，所以不用过多考虑性能的问题；另外，使用外键可以降低开发成本，借助数据库产品自身的触发器可以实现表与关联表之间的数据一致性和更新；最后一点，使用外键的方式，还可以做到开发人员和数据库设计人员的分工，可以为程序员承担更多的工作量；

为何说外键有性能问题：
1. 数据库需要维护外键的内部管理；
2. 外键等于把数据的一致性事务实现，全部交给数据库服务器完成；
3. 有了外键，当做一些涉及外键字段的增，删，更新操作之后，需要触发相关操作去检查，而不得不消耗资源；
4. 外键还会因为需要请求对其他表内部加锁而容易出现死锁情况；

首先我们明确一点，外键约束是一种约束，这个约束的存在，会保证表间数据的关系“始终完整”。因此，外键约束的存在，并非全然没有优点。
比如使用外键，可以

1、保证数据的完整性和一致性

2、级联操作方便

3、将数据完整性判断托付给了数据库完成，减少了程序的代码量

然而，鱼和熊掌不可兼得。外键是能够保证数据的完整性，但是会给系统带来很多缺陷。正是因为这些缺陷，才导致我们不推荐使用外键，具体如下

性能问题

假设一张表名为 user_tb。那么这张表里有两个外键字段，指向两张表。那么，每次往 user_tb 表里插入数据，就必须往两个外键对应的表里查询是否有对应数据。如果交由程序控制，这种查询过程就可以控制在我们手里，可以省略一些不必要的查询过程。但是如果由数据库控制，则是必须要去这两张表里判断。

并发问题

在使用外键的情况下，每次修改数据都需要去另外一个表检查数据,需要获取额外的锁。若是在高并发大流量事务场景，使用外键更容易造成死锁。

扩展性问题

这里主要是分为两点

做平台迁移方便，比如你从 Mysql 迁移到 Oracle，像触发器、外键这种东西，都可以利用框架本身的特性来实现，而不用依赖于数据库本身的特性，做迁移更加方便。

分库分表方便，在水平拆分和分库的情况下，外键是无法生效的。将数据间关系的维护，放入应用程序中，为将来的分库分表省去很多的麻烦。

技术问题

使用外键，其实将应用程序应该执行的判断逻辑转移到了数据库上。那么这意味着一点，数据库的性能开销变大了，那么这就对 DBA 的要求就更高了。很多中小型公司由于资金问题，并没有聘用专业的 DBA，因此他们会选择不用外键，降低数据库的消耗。
相反的，如果该约束逻辑在应用程序中，发现应用服务器性能不够，可以加机器，做水平扩展。如果是在数据库服务器上，数据库服务器会成为性能瓶颈，做水平扩展比较困难。

16.3 数据库的设计

创建 "商品分类" 表(之前已经创建,无需再次创建)

```
create table goods_cates(
    id int unsigned primary key auto_increment not null,
    name varchar(40) not null
);
```

创建 "商品品牌" 表(之前已经创建,无需再次创建)

```
create table goods_brands (
    id int unsigned primary key auto_increment not null,
    name varchar(40) not null
);
```

创建 "商品" 表(之前已经创建,无需再次创建)

```
create table goods(
    id int unsigned primary key auto_increment not null,
    name varchar(40) default '',
    price decimal(5,2),
    cate_id int unsigned,
    brand_id int unsigned,
    is_show bit default 1,
    is_saleoff bit default 0,
    foreign key(cate_id) references goods_cates(id),
    foreign key(brand_id) references goods_brands(id)
);
```

创建"顾客"表

```
create table customer(
    id int unsigned auto_increment primary key not null,
    name varchar(30) not null,
    addr varchar(100),
    tel varchar(11) not null
);
```

创建"订单"表

```
create table orders(
    id int unsigned auto_increment primary key not null,
    order_date_time datetime not null,
    customer_id int unsigned,
    foreign key(customer_id) references customer(id)
);
```

创建"订单详情"表

```
create table order_detail(
    id int unsigned auto_increment primary key not null,
    order_id int unsigned not null,
    goods_id int unsigned not null,
    quantity tinyint unsigned not null,
    foreign key(order_id) references orders(id),
    foreign key(goods_id) references goods(id)
);
```

说明

- 以上创建表的顺序是有要求的,即如果 goods 表中的外键约束用的是 goods_cates 或者是 goods_brands,那么就应该先创建这 2 个表,否则创建 goods 会失败
- 创建外键时,一定要注意类型要相同,否则失败

表的多对多关系

多对多

ID(Primary key)	teacher
1	chinese
2	math
3	english
4	c

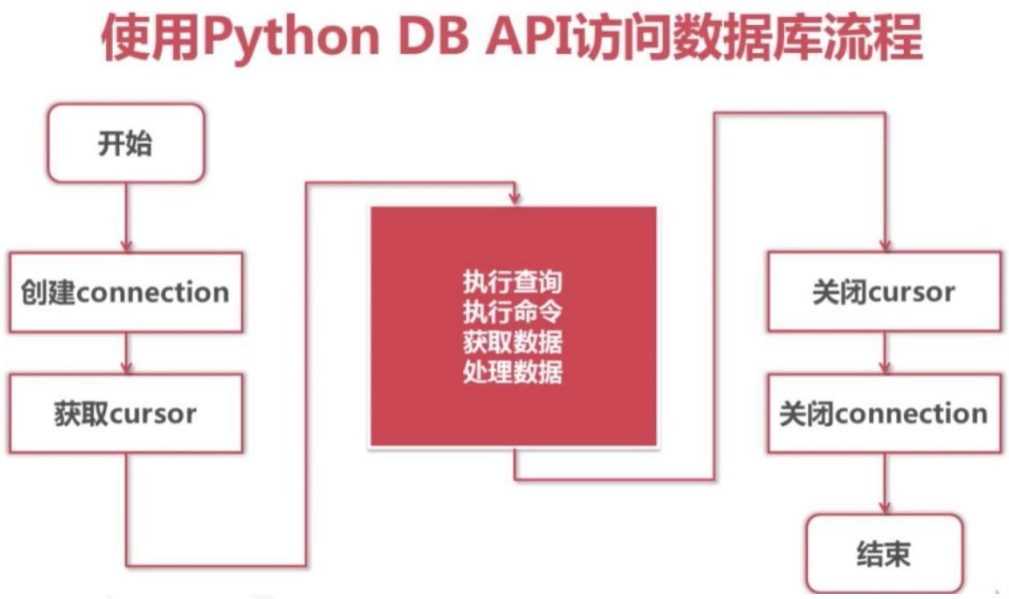
ID(PK)	Name
1	
2	
3	
4	

T_ID	S_ID
1	1
1	2
2	1
2	2
3	1

```
create table TEACHER(
ID int primary key,
NAME varchar(100)
);
insert into TEACHER values(1,'math teacher');
insert into TEACHER values(2,'chinese teacher');
insert into TEACHER values(3,'english teacher');

create table STUDENT3(
ID int primary key,
NAME varchar(100)
);
```

```
insert into STUDENT3 values(1,"lily");
insert into STUDENT3 values(2,"lucy");
insert into STUDENT3 values(3,"lilei");
insert into STUDENT3 values(4,"xiongda");


create table TEACHER_STUDENT(
T_ID int,
S_ID int,
primary key(T_ID,S_ID),
constraint T_ID_FK foreign key(T_ID) references TEACHER(ID),
constraint S_ID_FK foreign key(S_ID) references STUDENT3(ID));

insert into TEACHER_STUDENT VALUES(1,2);

select t.`NAME`,s.`NAME` from TEACHER as t INNER JOIN TEACHER_STUDENT as ts
on ts.T_ID=t.ID INNER JOIN STUDENT3 as s on s.ID=ts.S_ID;

select t.`NAME`,s.`NAME` from TEACHER as t LEFT JOIN TEACHER_STUDENT as ts
on ts.T_ID=t.ID LEFT JOIN STUDENT3 as s on s.ID=ts.S_ID;
```

16.4 Python 中操作 MySQL 步骤



引入模块

- 在 py 文件中引入 pymysql 模块

```
from pymysql import *
```

Connection 对象

- 用于建立与数据库的连接
- 创建对象：调用 connect()方法

```
conn=connect(参数列表)
```

- 参数 host：连接的 mysql 主机，如果本机是'localhost'

- 参数 port：连接的 mysql 主机的端口，默认是 3306
- 参数 database：数据库的名称
- 参数 user：连接的用户名
- 参数 password：连接的密码
- 参数 charset：通信采用的编码方式，推荐使用 utf8

对象的方法

- close()关闭连接
- commit()提交
- cursor()返回 Cursor 对象，用于执行 sql 语句并获得结果

Cursor 对象

- 用于执行 sql 语句，使用频度最高的语句为 select、insert、update、delete
- 获取 Cursor 对象：调用 Connection 对象的 cursor()方法

```
cs1=conn.cursor()
```

对象的方法

- close()关闭
- execute(operation [, parameters])执行语句，返回受影响的行数，**主要用于执行 insert、update、delete 语句**，也可以执行 create、alter、drop 等语句（**尽量避免 create，alter，drop 等操作，大公司由 DBA 进行**）
- **fetchone()执行查询语句时，获取查询结果集的第一个行数据，返回一个元组**
- **fetchall()执行查询时，获取结果集的所有行，一行构成一个元组，再将这些元组装入一个元组返回**

对象的属性

- rowcount 只读属性，表示最近一次 execute()执行后受影响的行数
- connection 获得当前连接对象

16.5 增删改

```
from pymysql import *

def main():
    # 创建 Connection 连接
    conn = connect(host='localhost',port=3306,database='jing_dong',user
='root',password='mysql',charset='utf8')
    # 获得 Cursor 对象
    cs1 = conn.cursor()
    # 执行 insert 语句，并返回受影响的行数：添加一条数据
    # 增加
    count = cs1.execute('insert into goods_cates(name) values("硬盘")')
    #打印受影响的行数
    print(count)

    count = cs1.execute('insert into goods_cates(name) values("光盘")')
    print(count)

    # # 更新
    # count = cs1.execute('update goods_cates set name="机械硬盘" where
name="硬盘"')
    # # 删除
    # count = cs1.execute('delete from goods_cates where id=6')

    # 提交之前的操作，如果之前已经之执行过多次的 execute，那么就都进行提交
    conn.commit()

    # 关闭 Cursor 对象
    cs1.close()
```

```
    # 关闭 Connection 对象
    conn.close()

if __name__ == '__main__':
    main()
```

查询一行数据

```
from pymysql import *

def main():
    # 创建 Connection 连接
    conn = connect(host='localhost',port=3306,user='root',password='mys
ql',database='jing_dong',charset='utf8')
    # 获得 Cursor 对象
    cs1 = conn.cursor()
    # 执行 select 语句，并返回受影响的行数：查询一条数据
    count = cs1.execute('select id,name from goods where id>=4')
    # 打印受影响的行数
    print("查询到%d 条数据:" % count)

    for i in range(count):
        # 获取查询的结果
        result = cs1.fetchone()
        # 打印查询的结果
        print(result)
        # 获取查询的结果

    # 关闭 Cursor 对象
    cs1.close()
    conn.close()

if __name__ == '__main__':
    main()
```

查询多行数据

```
from pymysql import *

def main():
    # 创建 Connection 连接
    conn = connect(host='localhost',port=3306,user='root',password='mys
ql',database='jing_dong',charset='utf8')
    # 获得 Cursor 对象
    cs1 = conn.cursor()
    # 执行 select 语句，并返回受影响的行数：查询一条数据
    count = cs1.execute('select id,name from goods where id>=4')
    # 打印受影响的行数
```

```
    print("查询到%d 条数据:" % count)

    # for i in range(count):
    #     # 获取查询的结果
    #     result = cs1.fetchone()
    #     # 打印查询的结果
    #     print(result)
    #     # 获取查询的结果

    result = cs1.fetchall()
    print(result)

    # 关闭 Cursor 对象
    cs1.close()
    conn.close()

if __name__ == '__main__':
    main()
```

当数据量非常多时，要用 fetchone，否则可能会造成 oom 错误

16.6 参数化

- sql 语句的参数化，可以有效防止 sql 注入
- 注意：此处不同于 python 的字符串格式化，全部使用%s 占位

```
from pymysql import *

def main():

    find_name = input("请输入物品名称：")

    # 创建 Connection 连接
    conn = connect(host='localhost',port=3306,user='root',password='mys
ql',database='jing_dong',charset='utf8')
    # 获得 Cursor 对象
    cs1 = conn.cursor()


    # # 非安全的方式
    # # 输入 " or 1=1 or "   (双引号也要输入)
    # sql = 'select * from goods where name=%s' % find_name
    # print("""sql===>%s<====""" % sql)
    # # 执行 select 语句，并返回受影响的行数：查询所有数据
    # count = cs1.execute(sql)

    # 安全的方式
    # 构造参数列表
    params = [find_name]
    # 执行 select 语句，并返回受影响的行数：查询所有数据
    count = cs1.execute('select * from goods where name=%s', params)
    # 注意：
    # 如果要是有多个参数，需要进行参数化
    # 那么 params = [数值 1, 数值 2....]，此时 sql 语句中有多个%s 即可

    # 打印受影响的行数
    print(count)
    # 获取查询的结果
    # result = cs1.fetchone()
    result = cs1.fetchall()
    # 打印查询的结果
    print(result)
    # 关闭 Cursor 对象
    cs1.close()
```

```
    # 关闭 Connection 对象
    conn.close()

if __name__ == '__main__':
    main()
```

数据库连接池：

100 个

为了实现多进程编程时，当没有连接可用时，进程就睡觉