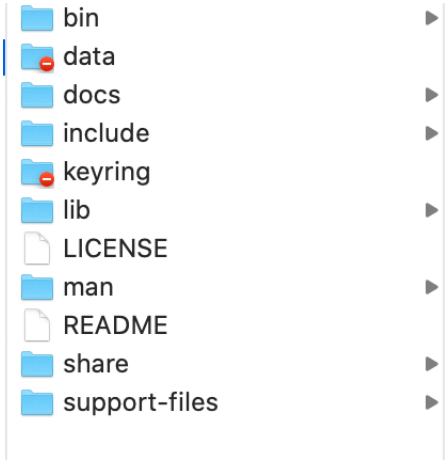
1.1 MySql基础，SQL入门

1.1.1 数据库的基本知识

数据库是存储和管理数据的仓库。

1.1.9 MySql的目录结构



1. bin用来存放可执行文件
2. data用来保存数据库和数据表的信息
3. docs用来存放文档
4. include用来存放头文件
5. lib用来存放依赖库
6. share用来存放字符集和语言

1.1.10 数据库管理系统

数据库管理系统 - Database Management System。MySql就是一个数据管理系统。

1.1.11 数据库表

1. 数据库中以表为单位存储数据
2. 一个表相当于java中一个类

1.1.12 SQL的概念

1. Structured Query Language
2. 用于管理关系型数据库的语言

1.1.13 SQL的通用语法

1. 可以单行或多行出现以分号（;）为结尾
2. 使用空格和缩进增加可读性
3. 不区分大小写
4. 注释方式
    - SHOW DATABASES; -- 注释
    - SHOW DATABASES; /*注释*/
    - SHOW DATABASES; #注释

1.1.14 SQL的分类

分类	说明
数据定义语言	DDL - 用来操作数据库和表的
数据操作语言	DML - 对表中的数据进行增删改的
数据查询语言	DQL - 查询表中的数据
数据控制语言	DCL - 定义数据库的访问权限，安全级别，创建用户

1.1.15 DDL-操作数据库-创建&查询

对数据库操作的分类：CRUD 和使用数据库

常用语句

语句	功能介绍
create database 数据库名;	这种方式采用默认编码集latin1，会导致插入数据乱码
create database 数据库名 character set 字符集编码;	
use 数据库名;	
select database();	查看当前正在使用的数据库
show databases;	查询mysql中有哪些数据库

1.1.16 mysql自带数据库的介绍

Information_schema : 信息数据库，保存其他的数据库信息的，比如数据库名称，表名称

mysql：mysql核心数据库，保存用户和权限信息

performance_schema：保存性能相关数据，监控mysql的性能

sys: 记录DBA所需的一些信息，更方便的让DBA快速了解数据库的运行情况的 （database assistance 数据库管理员）

1.1.17 DDL-操作数据库-修改&删除

语句	功能介绍
alter database 数据库名 character set 字符集；	修改数据库的字符集
show create database 数据库名;	查询当前数据库的信息
drop database 数据库名;	删除数据库

1.1.18 MySql的常见数据类型

数据类型	
int	
double	
date	日期类型 只显示年月日 yyyy-MM-dd
datetime	年月日时分秒 yyyy-MM-dd HH:mm:ss
char	字符串：是固定长度，指定了多少长度，创建时就开辟多少空间
varchar	字符串：可变长度的类型，存储字符串时，只使用所需的空间

1.1.19 DDL-操作数据表-创建&查看

语句	功能介绍
create table 表名 (字段名称1 字段类型, 字段名称2 字段类型);	
create table 新表名 like 旧表名	快速创建一个表结构相同的表
desc 表名	查看表结构
show tables;	查看当前数据库中所有的表名
show create table 表名	查看创建表的sql

1.1.20 DDL-操作数据表-删除

语句	功能介绍
drop table 表名;	
drop table if exists 表名;	

1.1.21 DDL-操作数据表-修改

语句	功能介绍
rename table 旧表名 to 新表名;	修改表的名称
alter table 表名 character set 字符集;	修改表的字符集
alter table 表名 add 字段名称 字段类型(长度);	添加表中的某一列 关键字 add
alter table 表名 modify 字段名称 字段类型(长度);	修改表中列的类型或长度 关键字 modify
alter table 表名 change 旧列名 新列名 类型(长度);	修改表中列的名称 关键字 change
alter table 表名 drop 列名;	删除某一列 关键字 drop

1.1.22 DML-插入数据

```
create TABLE student(
  sid int,
  sname varchar(20),
  age int,
  sex char(1),
  address varchar(40)
);
```

语句	功能介绍
insert into 表名 (sid, sname,age,sex,address) values (1, "zichen", 27, "男", "McLean");	
insert into 表名 values (1, "zichen", 27, "男", "McLean");	
insert into (sid, sname) 表名 values (1, "zichen");	插入指定字符

注意⚠️:

1. 在插入char，varchar，date类型时，必须加" " 或者 ‘’
2. 如果插入空值可以不写，或者写null

1.1.23 DML-修改操作

语句	功能介绍
update 表名 set 列名 = 值;	
update 表名 set 列名 = 值 where (条件表达式：字段名称 = 值);	
update 表名 set 字段名称1 = 值， 字段名称2 = 值 where (条件表达式：字段名称= 值);	一次性修改多个列

1.1.24 DML-删除数据

语句	功能介绍
delete from 表名;	不推荐 对表中的数据逐条删除，效率低
truncate table 表名;	推荐，删除整张表，然后在创建一个一摸一样的新表
delete from 表名 where (条件表达式：字段名称 = 值);	

1.1.25 DQL-简单查询

```
create table emp(
  eid int,
  ename varchar(20),
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20)
);
```

语句	功能介绍
select * from 表名;	选择表中所有信息
select 列名 from 表名;	选择某一列的所有信息
select 列名1, 列名2 from 表名;	选择指定列的所有信息
select eid as '编号', ename as '姓名', sex as '性别', hire_date as '入职日期', dept_name as '部门信息' from emp	查询所有的数据，然后给列名改为中文
select distinct dept_name from emp;	去重操作 关键字distinct
select ename, salary+1000 as salary from emp;	将工资 +1000 然后显示

注意⚠️：查询操作不会对数据表中的数据进行修改，只是一种显示的方式

1.1.26 DQL-条件查询

语句	功能介绍
select 列名 from 表明 where 条件表达式	

比较运算符

运算符	功能介绍
>, <, <=, >=, =, != <>	
between ... and ...	例如：2000～10000： between 2000 and 10000
in(集合)	例如：name in (悟空，八戒)
like '%张%'	模糊查询，只要包含张字的就可以查询出来。**%通配符，表示任意多个字符**
like '%张'	以张结尾
like '张%'	以张开头
_	表示一个字符
is null	查询某一列为null的值

逻辑运算符

运算符	功能介绍
and &&	
or ||	
not	

注意⚠️：先取出表中的每条数据，满足条件就返回，不满足就过滤掉。

```
select * from emp where ename = "黄蓉";

select * from emp where salary = 5000;

select * from emp where salary != 5000;

select * from emp where salary > 6000;

select * from emp where salary between 5000 and 10000;

select * from emp where salary = 3600 or salary = 7200 orsalary = 20000;

select * from emp where salary in(3600, 7200, 20000);



select * from emp where ename like '%精%';
```

```sql
select * from emp where ename like '孙%';

select * from emp where ename like '_兔%';

select * from emp where dept_name is null;

select * from emp where dept_name is not null;
```

1.2 MySql单表，约束和事务

1.2.1 DQL-排序查询

```sql
create table emp(
  eid int,
  ename varchar(20),
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20)
);
```

语句	功能介绍
select 字段名称 from 表名 [where 字段名 = 值] order by 字段名称 [asc/desc]	默认升序排序
select * from emp order by salary desc, eid desc	薪资相同时，按eid排序

1.2.2 DQL-聚合函数

将一列数据作为整体作为一个整体，进行纵向的计算的

常用的聚合函数

聚合函数	功能介绍
count(字段名称)	统计记录数，会忽略空值。不要使用空值的列进行统计
sum(字段名称)	
max(字段名称)	
min(字段名称)	
avg(字段名称)	

语句	功能介绍
select 聚合函数(字段名称) from 表名 [where 条件]	

```
create table emp(
  eid int,
  ename varchar(20),
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20)
);
```

```
select count(*) from emp;

select count(1) from emp;

select count(eid) from emp;

select sum(salary) as '总薪水', max(salary) as '最高薪水', min(salary) as '最小薪水',
avg(salary) as '平均薪水'  from emp;

select count(*) from emp where salary > 4000;

select count(*) from emp where dept_name = '教学部';

select  avg(salary) from emp where dept_name = '市场部';
```

1.2.3 DQL-分组查询

分组的目的是为了做统计操作，一般分组会和聚合函数一起使用。

语句	功能介绍
select 分组字段/聚合函数 from 表名 group by 分组字段	

select * from emp group by sex;

1. 将性别相同的数据分为一组
2. 返回的是每组的第一条数据

```
select sex, avg(salary) from emp group by sex;

select dept_name as '部门名称' from emp group by dept_name;

select dept_name, avg(salary) from emp group by dept_name;

select dept_name, avg(salary) from emp where dept_name is not null group by
dept_name;
```

在分组之后进行条件过滤，使用having

```
select dept_name, avg(salary) from emp where dept_name is not null group by
dept_name having avg(salary) > 6000;
```

注意⚠️ where 与 having区别:

where

1. 在分组前进行过滤
2. where后不能接聚合函数

having

1. 在分组后进行过滤
2. hivinv后面可以接聚合函数

1.2.4 limit 关键字

通过limit指定要查的数据的条数

语句	功能介绍
select 字段名称 from 表名 limit offset, length;	offset：起始函数，默认从何0开始；length：返回的行数

```
select * from emp limit 0, 5;

select * from emp limit 5;

select * from emp limit 3, 6;
```

分页公式：起始行数 = (当前页码 - 1)*每页显示行数

1.2.5 约束的介绍

约束是指对数据进行一定的限制，来保证数据的完整性，有效性，正确性

常见约束	
主键约束	关键字：primary key；不可重复，唯一，非空；作用：用来表示数据库中的每一条记录
唯一约束	关键字：unique；表中的某一列不能重复，对null值不做唯一判断
非空约束	关键字：not null；某一列不许为空
外键约束	

1.2.6 主键约束

```
create table emp(
  eid int primary key, -- 指定eid为主键
  ename varchar(20),
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20)
);
```

```
create table emp(
  eid int,
  ename varchar(20),
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20),
  primary key(eid) -- 指定eid为主键
);
```

```
create table emp(
  eid int,
  ename varchar(20),
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20),
  primary key(eid)
);

alter table emp add primary key(eid); -- 指定eid为主键
```

```
alter table emp drop primary key; --删除主键
```

1.2.7 主键自增

关键字：auto_increment；字段类型必须是整数类型

```
create table emp(
  eid int primary key auto_increment, -- 主键自增
  ename varchar(20),
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20)
);
```

对主键起始值的修改

```
create table emp(
  eid int primary key auto_increment, -- 主键自增
  ename varchar(20),
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20)
) auto_increment = 100; -- 主键起始值为100
```

1.2.8 Delete和Truncate对自增的影响

关键字	功能介绍
Delete	逐条删除；对自增没有影响
Truncate	将整个表删除，然后创建一个结构相同的表；自增从初始值开始

1.2.9 非空约束

```
create table emp(
  eid int primary key, -- 指定eid为主键
  ename varchar(20) not null, -- 非空约束
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20)
);
```

1.2.10 唯一约束

```
create table emp(
  eid int primary key, -- 指定eid为主键
  ename varchar(20) unique, -- 唯一约束
  sex char(1),
  salary double,
  hire_date date,
  dept_name varchar(20)
);
```

注意⚠️：主键约束和唯一约束的区别

主键约束：

1. 唯一且不能为空

2. 一个表中只有一个主键约束

唯一约束：

1. 唯一，但可以为空
2. 一个表中可以有多个唯一约束

1.2.11 默认值

关键字：default；用来指定某一列的默认值

```
create table emp(
  eid int primary key, -- 指定eid为主键
  ename varchar(20) unique, -- 唯一约束
  sex char(1) default '女', -- 默认值女
  salary double,
  hire_date date,
  dept_name varchar(20)
);
```

1.2.12 事务的基本概念&转账操作

基本概念

事务：一个由一条或多条sql语句组成的整体，事务中的操作，要么全部成功，要么全部失败。

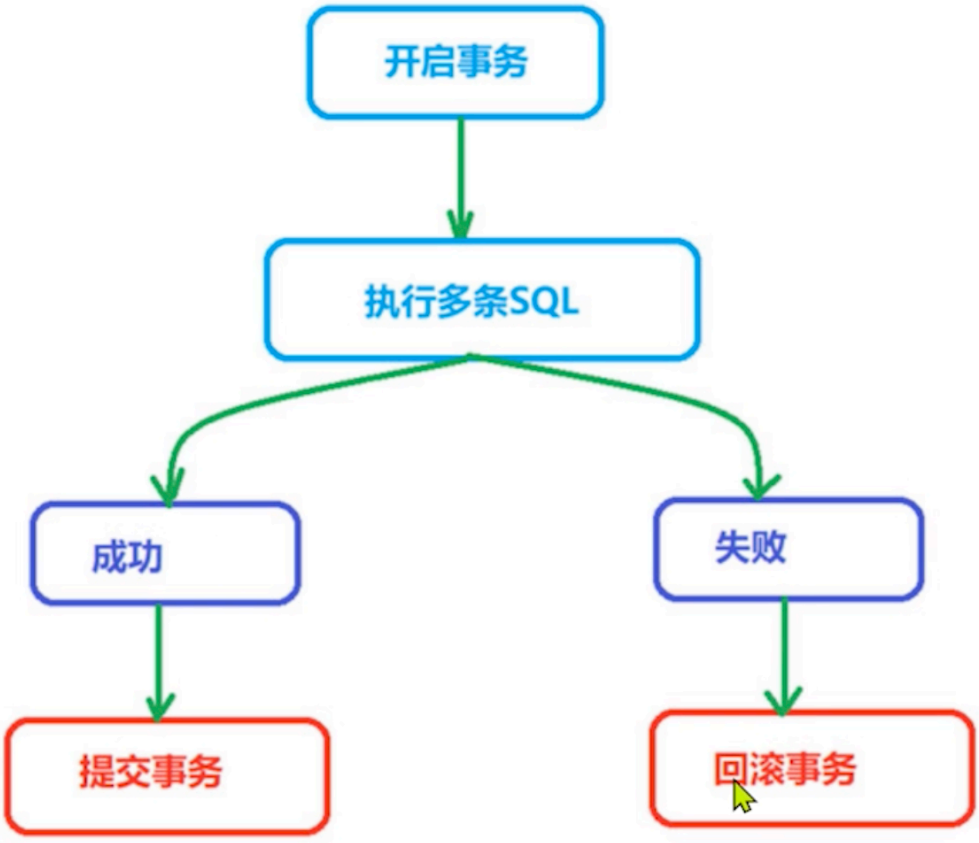
回滚：撤销执行的操作

账户表

主键	账户名	金额
1	张三	1000
2	李四	1000

1.2.13 MySql手动增加事务

开启事务	start transaction；或者 begin
提交事务	commit
回滚事务	rollback，出现异常时，回到事务开启之前的状态



1.2.14 MySql自动增加事务

自动提交事务是默认的提交方式。每执行一条DML语句都一个单独的事务。

```
show variables like 'autocommit';
set @@autocommit = off;
```

1.2.15 MySql的四大特性

1. 原子性：每个事务都是一个整体，不可以拆分，事务中的所有sql要么全部成功，要么全部失败
2. 一致性：事务在执行前，数据库的状态，与事务执行之后的状态要保持一致性
3. 隔离性：事务与事务之间不应该相互影响，执行时要保证隔离性
4. 持久性：一旦事务执行成功，对数据的修改是持久的

1.2.16 MySql事务隔离级别

各个事务之间是隔离的，相互独立的，但是多个事务对数据库中的同一批数据进行**并发访问**时，就会引起一些问题。

可以通过**设置不同的隔离级别**来解决对应的问题。

并发访问：

1. 脏读：一个事务读取到另一个事务没有提交的数据
2. 不可重复读：一个事务中，两次读取的数据不一致
3. 幻度：一个事务中，一次查询的结果无法支撑后续的业务操作

隔离级别

1. read uncommited: 不能防止任何问题
2. read commited: 可以防止脏读 (Oracle默认隔离级别)
3. repeatable read：可以防止脏读，不可重复读 (MySql 默认隔离级别)
4. serializable：可以防止脏读，不可重复读，幻读

隔离级别从小到大安全性越来越高，但效率越来越低。

1.2.17 隔离级别的相关命令

语句	功能介绍
select @@transaction_isolation;	查看隔离级别
set global transaction isolation level 级别名称	设置隔离级别

1.2.18 脏读演示及解决

脏读：一个事务读取到另一个事务没有提交的数据

1.2.19 不可重复读演示及解决

不可重复读：一个事务中，两次读取的数据不一致

1.2.20 幻读演示及解决

幻度：一个事务中，一次查询的结果无法支撑后续的业务操作

1.3 MySql多表，外键和数据库设计

1.3.1 多表的概述

1.3.2 外键约束

可以让两张表之间产生对应的关系，从而保证主从表的完整性。

外键：外键指的是在**从表**中与**主表**中的**主键**对应的字段

主表：主键id所在的表，一的一方

从表：外键所在的表，多的一方

```
create table employee(
  eid int primary key auto_increment,
  ename varchar(20),
  age int,
  dept_id int, -- 外键字段 指向主表的主键
  constraint emp_dept_fk foreign key(dept_id) references department(id) -- 添加外键
约束
);
```

```
create table department(
  id int primary key auto_increment,
  dept_name varchar(30),
  dept_location varchar(30)
);
```

添加外键约束之后，就会产生一个强制的外键约束检查，保证数据的完整性和一致性。

语法	
alter table 从表 add constaint 外键约束的名称 foreign key (外键字段) references 主表(主键)	

```
alter table employee add constraint emp_dept_fk foreign key(dept_id) references
department(id)
```

1.3.3 删除外键约束&外键注意事项

语句	功能介绍
alter table 从表 drop foreign key 外键约束的名称	

```
alter table employee drop foreign key emp_deppt_fk;
```

注意⚠️：

1. 从表的外键必须和主表的主键类型一致
2. 添加数据时应该先添加主表的数据，在添加从表的数据
3. 删除数据的时候，先删除从表的数据

1.3.4 级联删除

指的是在删除主表的数据的同时，可以删除与之相关的从表的数据

语句	功能介绍
on delete cascade	级联删除

```
create table employee(
  eid int primary key auto_increment,
  ename varchar(20),
  age int,
  dept_id int, -- 外键字段 指向主表的主键
  constraint emp_dept_fk foreign key(dept_id) references department(id), -- 添加外
键约束
  on delete cascade -- 级联删除
);
```

1.3.5 多表关系介绍

表与表之间的三种关系：

1. 一对多关系(1:n)：班级和学生；部门和员工
2. 多对多关系(n:n)：学生和课程；演员和角色
3. 一对一关系(1:1)：身份证和人

1.3.6 一对多关系介绍

建表原则：在多的一方建立外键，指向一的一方的主键

1.3.7 多对多关系介绍

建表原则：多对多的关系中，需要创建第三张表作为中间表。中间表中至少有两个字段，是两张表中的主键字段，作为中间表的外键。

1.3.8 一对一关系介绍

建表原则：可以在任意一方建立外键指向另一方的主键

1.3.9 设计省市表(一对多)

```
create table province(
  id int primary key auto_increment,
  name varchar(20),
  description varchar(20)
);
```

```
create table city(
  cid int primary key auto_increment,
  name varchar(20),
  description varchar(20),
  province_id int, -- 外键名称
  foreign key(province_id) references province(id) -- 创建外键
);
```

1.3.10 设计演员与角色表(多对多)

```
create table actor(
  id int primary key auto_increment,
  name varchar(20)
);
```

```
create table role(
  id int primary key auto_increment,
  name varchar(20)
);
```

```
create table actor_role(
  id int primary key auto_increment,
  aid int,
  rid int
);

alter table actor_role add foreign key(aid) references actor(id);
alter table actor_role add foreign key(rid) references role(id);
```

1.3.11 多表查询的介绍

1. 内链接查询
2. 外链接查询

语句	功能介绍
select 字段列表 from 表名列表；	

```
create table category(
  cid varchar(32) primary key,
  cname varchar(50)
);
```

```
create table products(
  pid varchar(32) primary key,
  pname varchar(50),
  price int,
  flag varchar(2), -- 1表示上架，0表示下架
  category_id varchar(32),
  foreign key(category_id) references category(cid)
);
```

笛卡尔积：交叉连接查询的结果会产生笛卡尔积，没办法使用

```
select * from products, category;
```

1.3.12 内链接查询

通过指定的条件去匹配俩张表中的数据，匹配不上的就不显示

1. 隐式内连接

语句	
select 字段名... from 表名.. where 连接条件	

```
select * from products, category where category_id = cid;

select p.pname, p.price, c.category from products p, category c where
p.category_id = c.cid;

select p.pname c.cname from products p, category c where p.category_id = c.cid
and p.pname = '格力空调';
```

2. 显示内连接

语法	
select 字段名... from 左表[inner] join 右表 on 连接条件	

```
select * from products p join category c on p.category_id = c.cid;

select
  p.pname,
  p.price
from products p join category c on p.category_id = c.cid
where c.cname='鞋服' and p.price > 500;
```

1.3.13 外链接查询

1. 左外连接

   以左表为基准，匹配右表上的数据，如果匹配上就显示，如果匹配不上，左表的数据正常显示，右表数据显示null。

语法	
select 字段名... from 左表 left join 右表 on 连接条件	

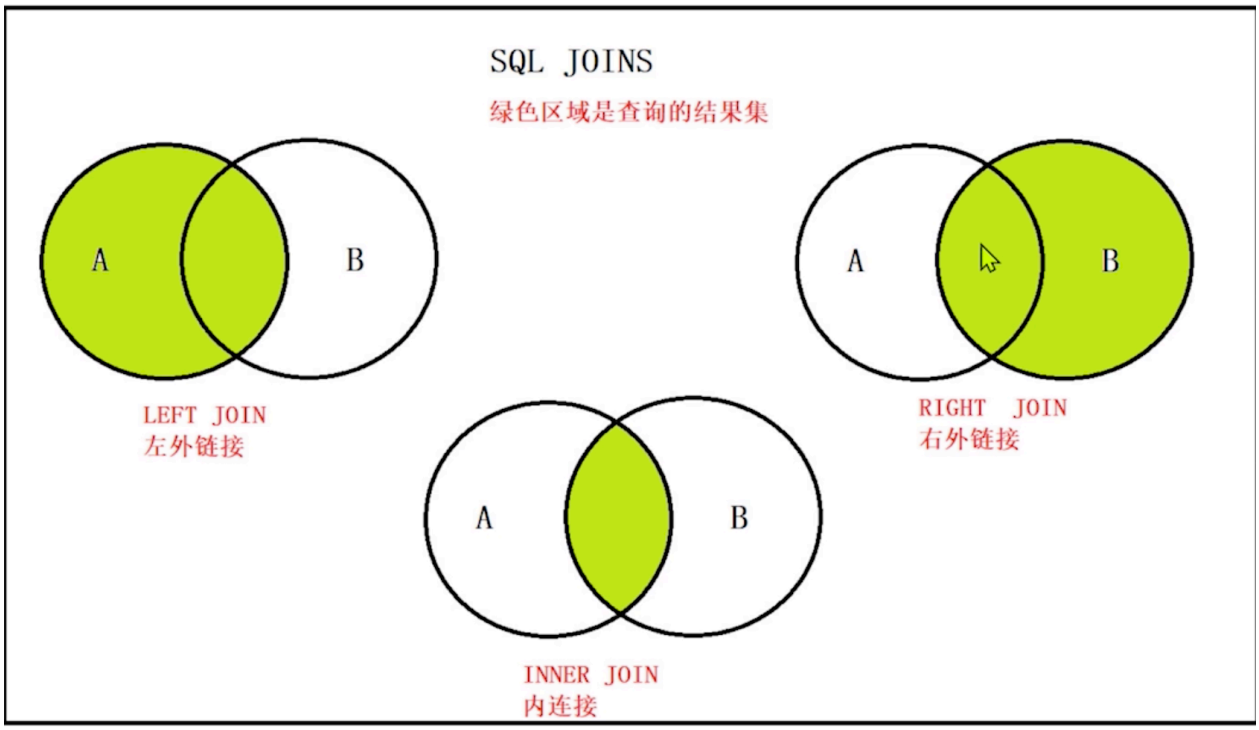
```
select
  c.cname,
  count(p.pid)
from category c left join products p on p.category_id = c.cid
group by c.cname;
```

2. 右外连接

   以右表为基准，匹配左表上的数据，如果匹配上就显示，如果匹配不上，右表的数据正常显示，左表数据显示null。

语句	
select 字段名... from 左表 right join 右表 on 连接条件	

```
select * from products p right join category c on p.category_id = c.cid
```



1.3.14 子查询

一条select语句的结果作为另外一条select语句的一部分。子查询必须要放在括号里。

```
select max(price)from products;

select * from products where price = 5000;

select * from products where price = (select max(price)from products);
```

1.3.15 子查询作为查询条件

子查询的分类：

1. where型子查询：将子查询的结果作为父类查询的比较条件使用
2. from型子查询：将子查询的查询结果最为一张表使用
3. exists型子查询：查询结果是单列多行的情况，可以将子查询的结果作为父查询的in函数中的条件使用

```
select cid from category where cname = '化妆品';

select
  p.pname,
  p.price
from products p
where p.category = (select cid from category where cname = '化妆品');
```

```
select avg(price) from products;

select
  *
from products
where price < (select avg(price) from products);
```

1.3.16 子查询作为一张表

```
select * from category;

select
  p.pname,
  p.price,
  c.cname
from products p
inner join (select * from category) c on p,category_id = c.cid
where p.price > 500;
```

1.3.17 子查询结果是单列多行

作为父查询的in函数中的条件使用

```
select distinct category_id from products where price < 2000;

select
  *
from category
where cid
in (select distinct category_id from products where price < 2000);
```

```
select cid from category where cname in ('家电','鞋服');

select
  *
from products where category_id
in (select cid from category where cname in ('家电','鞋服'));
```

总结：

1. 子查询如果是一个字段，那么就在where后面使用
2. 子查询如果是多行多列，就当一张表使用

1.3.18 数据库设计三范式

三范式值的是数据库设计的一种规则。

目的：创建冗余较小，结果合理的数据库。

第一范式(1NF)：满足最低要求的规范

第二范式(2NF)：在满足第一范式的基础之上，进一步满足更多的规范

第三范式(3NF)：以此类推

第一范式(1NF)

列具有原子性，设计列时要求做到列不可查分

第二范式(2NF)

一张表只能描述一件事情

第三范式(3NF)

消除传递信息：表中的信息如果能够被推导出来就不要在设计一个字段单独记录。**空间最省原则**

1.3.19 反三范式

通过增加冗余或重复数据来提高数据库的读性能。浪费存储空间节省查询时间(空间换时间)。

冗余字段：某一字段属于一张表，但是它又在多张表中都有出现。

总结：

1. 尽量遵循三范式
2. 合理的加入冗余字段，减少join操作

1.4 MySql索引，存储过程和触发器

1.4.1 索引的介绍

可以通过对数据表中的字段创建索引来提高查询速度

常见索引的分类：

1. 主键索引(primary key) 主键是一个唯一性的索引，每个表中只能有一个主键。
2. 唯一索引(unique) 索引列的所有数据只能出现一次
3. 普通索引(index) 最常见的索引，作用就是提高对数据的访问速度

表对应的索引被保存在一个索引文件中，如果对数据进行增删改查，那么mysql就需要对索引进行更新

1.4.2 索引的创建&删除

```
create table demo01(
  did int,
  dname varchar(20),
  hobby varchar(30)
);
```

主键索引

1. 创建表的时候直接添加主键
2. 创建表之后，添加索引

语句	
alter table 表名 add primary key(列名);	

唯一索引

1. 创建表的时候直接添加唯一索引
2. 创建表之后，使用crate添加索引

语句	
create unique index 索引名 on 表名(列名);	

3. 使用alter创建唯一索引

语句	
alter table 表名 add unique (列名);	

普通索引

1. 使用create方式创建索引

语句	
create index 索引名 on 表名(列名);	

2. 使用alter方式创建索引

语句	
alter table 表名 add index 索引名(列名);	

```
alter table demo01 add index idx_dname(dname);
```

删除索引

语句	
alter table 表名 drop index 索引名称;	

1.4.4 索引的优缺点

创建原则

优先选择为经常出现在查询条件或者排序分组后面的字段创建索引

优点

1. 提高查询速度
2. 减少查询中分组和排序的时间
3. 通过创建唯一索引保证数据的唯一性

缺点

1. 创建和维护索引需要索引时间，数据越大，时间越长
2. 表中的数据进行增删改查时，索引也需要进行维护，降低了维护的速度
3. 索引文件需要占据磁盘空间

1.4.5 视图的介绍和创建

基本概念

1. 视图是一种虚拟的表
2. 视图建立在已有表的基础之上，视图赖以建立的这些表称为基表
3. 向视图提供数据内容的语句为select语句，可以将视图理解为存储起来的select语句
4. 视图向用户提供基表数据的另一种表现形式

总结：视图是由查询结果形成的一张虚拟的表。主要是读数据，不会在视图上进行增删改操作

视图的作用

如果某个查询结果出现的十分频繁，并且查询语法比较复杂。那么这个时候，就可以根据这条查询语句构建一张视图，来方便查询。

创建视图

语句	
create view 视图名[字段列表] as select 查询语句;	

view：视图

字段列表：一般跟后面的查询语句相同

as select 查询语句：表示给视图提供数据的查询语句

```
create table category(
  cid varchar(32) primary key,
  cname varchar(50)
);
```

```
create table products(
  pid varchar(32) primary key,
  pname varchar(50),
  price int,
  flag varchar(2), -- 1表示上架，0表示下架
  category_id varchar(32),
  foreign key(category_id) references category(cid)
);
```

```
select * from products p left join category c on p.category_id = c.id;

create view products_category_view as
select * from products p left join category c on p.category_id = c.id;

select * from products_category_view;
```

1.4.6 使用视图进行查询操作

```
select
  c.cname,
  avg(p.price)
from products p left join category c on p.category_id = c.cid
group by c.cname;
```

```
select
  pc.cname,
  avg(pc.price)
from products_category_view pc
group by pc.cname;
```

```
select
 max(p.price)
from products p left join category c on p.category_id = c.cid
where c.cname='鞋服';

select
*
from products p left join category c on p.category_id = c.cid
where c.cname='鞋服' and p.price =
(
  select
```

```
    max(p.price)
  from products p left join category c on p.category_id = c.cid
  where c.cname='鞋服';
);
```

```
select
*
from products_category_view pc where pc.cname = '鞋服' and pc.price =
(
  select
  max(pc.price)
  from products_category_view pc
  where pc.cname = '鞋服
);
```

1.4.7 视图与表的区别

1. 视图是建立在表的基础之上的
2. 不要通过视图进行增删改操作
3. 删除视图表不受影响，如果删除表，视图就不再起作用了

1.4.8 存储过程的介绍

就是一堆sql语句的合并，中间加入了一些逻辑控制。像一个方法。

优点：

1. 调试完成可以稳定运行
2. 较少业务系统与数据库的交互

缺点

1. 互联网项目中，较少使用存储工程，因为业务需求变化太快
2. 存储工程移植十分困难

1.4.9 存储过程创建方式1

```
create table goods(
  gid int,
  name varchar(20),
  num int
);

create table orders(
  oid int,
  gid int,
  price int
);
```

语句	
delimiter `$$` create procedure 存储过程名称() begin --要执行的sql语句 end `$$`	

```
delimiter $$
create procedure goods_proc()
begin
  select * from good;
end $$

call goods_proc;
```

1.4.10 存储过程创建方式2

语法	
delimiter `$$` create procedure 存储过程名称(in 参数名 参数类型) begin --要执行的sql语句 end `$$`	

```
delimiter $$
create procedure goods_proc02(in goods_id int)
begin
  delete from goods where gid = goods_id;
end $$

call goods_proc02(1);
```

1.4.11 存储过程创建方式3

语句	功能介绍
set @变量名 = 值	变量的赋值
out 变量名 数据类型	out输出参数

```
delimiter $$
create procedure orders_proc(in o_oid int, in o_gid int, in o_price int, out
out_num int)
begin
  insert into orders values(o_oid, o_gid, o_price);
  set@out_num = 1;
  select @out_num;
end $$

call orders_proc(1, 2, 50, @out_num);
```

1.4.12 触发器的介绍

当我们执行一条sql语句的时候，这条sql语句的执行就会自动触发执行其他的sql语句。

四个要素

1. 监视地点(table)
2. 监视事件(insert/update/delete)
3. 触发时间(before/after)
4. 触发事件(insert/update/delete)

1.4.13 触发器的创建及使用

语句	
delimiter $ create trigger 触发器名 after/before (insert/update/delete) on tableName for each row begin --要执行的sql语句 end $	

```
delimiter $
create trigger t1
after insert
on orders
for each row
begin
  update goods set nun = num - 1 where gid = 4;
end $

insert into orders values(1,4,25)
```

1.4.14 DCL创建用户

语句	
create user '用户名'@'主机名' identified by '密码';	

```
create user 'admin1'@'localhost' identified by '123456';

create user 'admin2'@'%' identified by '123456'; --任意电脑上都可以登录
```

1.4.15 DCL用户授权

语句	
grant 权限1, 权限2 ... on 数据库名.表名 to '用户名'@'主机名';	

```
grant select on db4.products to 'admin1'@'localhost';
grant all on *.* to 'admin2'@'%';
```

1.4.16 DCL查看用户权限

语句	
show grants for '用户名'@'主机名'	

1.4.17 DCL查询用户&删除用户

语句	
drop user '用户名'@'主机名';	

1.4.19 数据库的备份-命令行方式

语句	
mysqldump -u 用户名 -p 密码 数据库名 > 文件路径	备份
source sql文件地址	还原

1.5 JDBC

1.5.1 JDBC概述

我们希望有一种统一的方式去操作所有的关系型数据库。JDBC是访问关系型数据库的标准规范(接口)，具体的实现方式由数据库厂商完成。

不同的数据库厂商提供了不同的实现类 -- 数据库驱动

程序员想要访问某个数据库时，只需要使用数据库驱动jar包。程序员只需要面向接口编程。

总结：

1. JDBC是访问关系型数据库的标准规范(接口)
2. 数据库厂商需要实现这套接口，并提供数据库驱动jar包
3. 我们去使用这套接口，真正执行的是对应的驱动包中实现类

1.5.3 JDBC开发-注册驱动

```
Class.for("com.mysql.jdbc.Driver");
```

```
Driver类是由MySql驱动包提供的一个实现类 他实现了 java.sql.Driver
public class Driver extends NonRegisteringDriver implements java.sql.Driver {

    public Driver() throws SQLException {
    }
    静态代码块 随着类的加载而加载 只加载一次
    static {

        try {
            DriverManager 类就是 驱动管理类 registerDriver 方法就是用来注册驱动的
            DriverManager.registerDriver(new Driver());
        } catch (SQLException var1) {
            throw new RuntimeException("Can't register driver!");
        }
    }
}
```

1.5.4 JDBC开发-获取连接

语句	功能介绍
DriverManager.getConnection(String url, String user, String password)	获取连接。url：mysql的URL格式，jdbc:mysql://ip地址:端口号/数据库名；user：登录用户名；password：登录密码

1.5.5 JDBC开发-获取语句执行对象

通过Connection的creatStatement方法获取sql语句执行对象

方法	功能介绍
Statement createStatement()	创建sql语句执行对象

Statement类

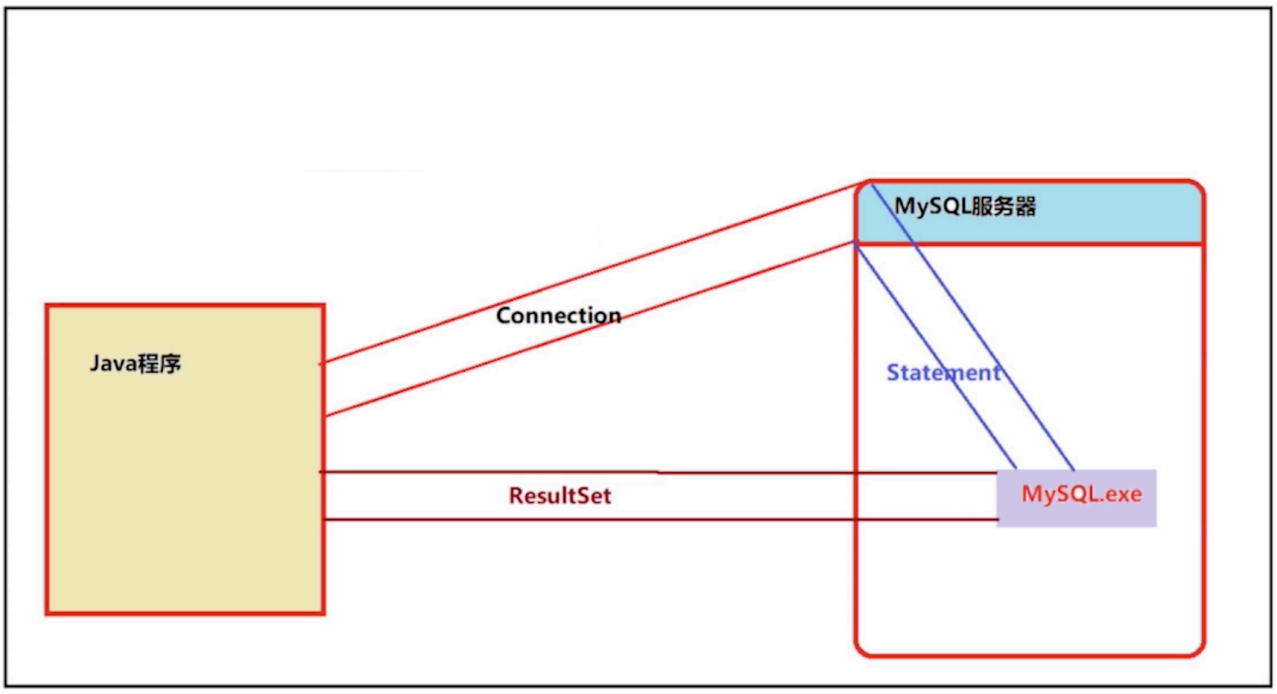
方法	功能介绍
int executeUpdate(String sql)	执行insert，update，delete语句。返回int类型，代表受影响的行数。
ResultSet executeQuery(String sql)	执行select语句，返回ResultSet结果集对象。

1.5.6 JDBC开发- 处理结果集对象

ResultSet接口

方法	功能介绍
boolean next()	判读是否有下一个数据
xxx getXxx(String or int)	如果传递的是int类型，表示通过列号查询；如果传递的是string类型，表示通过列名查询。

1.5.7 JDBC开发- 释放资源



1.5.11 SQL注入问题演示

用户输入的内容与我们编写的SQL进行拼接，进而导致了一些安全问题。

1.5.13 使用预处理防止SQL注入

不能让用户输入的内容和我们编写的SQL进行拼接

PrepareStatement接口

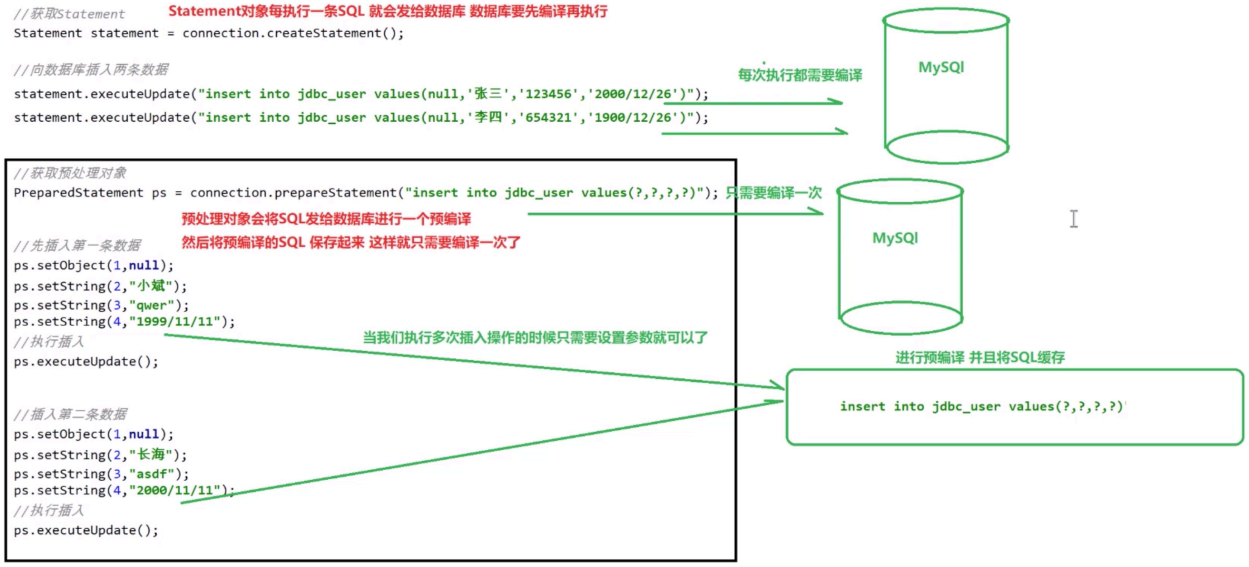
1. PrepareStatement接口是statement接口的子接口
2. 它有预编译的功能，提高sql的执行效率
3. 通过占位符的方式，设置参数可以有效的防止sql注入

语句	功能介绍
PreparedStatement preparedStatement(String sql)	使用？作为占位符
setXxx(占位符的位置, 要设置的值)	设置参数

1.5.14 预处理对象的执行原理

Statement对象每执行一条sql就会发给数据库，数据库要先编译再执行

PreparedStatement对象会将sql发给数据库进行一个预编译，然后将预编译的sql保存起来，这样就只要编译一次。当我们执行多次插入操作的时候，只需要设置参数即可。



1.5.15 JDBC控制事务API介绍

方法	功能介绍
void setAutocommit(boolean autoCommit)	如果是false，表示关闭自动提交，相当于开启事务。
void commit()	提交事务
void rollback()	回滚事务

1.5.16 JDBC控制事务-代码演示

1.6 数据库连接池和DBUtils

1.6.1 数据库连接池介绍

连接池技术管理数据库连接，可以重复使用连接。关闭连接不代表销毁了connection。只是将连接进行了归还。

1.6.2 如何使用连接池

Java提供了一个公共接口：javax.sql.DataSource。各个厂商需要让自己的连接池实现这个接口。

常用的连接池：DBCP，C3P0， Druid

1.6.3 DBCP连接池介绍

DBCP是开源的连接池。实现类BasicDataSource类。

1.6.4 DBCP连接池工具类编写

```
public class DBCPUtils {

    public static final String DRIVERNAME = "com.mysql.cj.jdbc.Driver";
    public static final String URL = "jdbc:mysql://localhost:3306/db1?
characterEncoding=UTF-           8&serverTimezone=UTC";
    public static final String USER = "root";
    public static final String PASSWORD = "Fzc1234!";

    public static BasicDataSource dataSource = new BasicDataSource();

    static {
        dataSource.setDriverClassName(DRIVERNAME);
        dataSource.setUrl(URL);
        dataSource.setUsername(USER);
        dataSource.setPassword(PASSWORD);
    }

    public static Connection getConnection(){
        Connection connection = null;
        try {
            connection = dataSource.getConnection();
        } catch (SQLException exception) {
            exception.printStackTrace();
        }
        return connection;
    }

    public static void close(Connection con, Statement statement){
        if(con != null && statement != null){
```

```
            try {
                statement.close();
                con.close();
            } catch (SQLException exception) {
                exception.printStackTrace();
            }
        }
    }

    public static void close(Connection con, Statement statement, ResultSet
resultSet){
        if(con != null && statement != null && resultSet != null){
            try {
                resultSet.close();
                statement.close();
                con.close();
            } catch (SQLException exception) {
                exception.printStackTrace();
            }
        }
    }
}
```

1.6.6 DBCP常见的配置介绍

属性	功能介绍
driverClassName	
url	
username	
password	
maxActive	
maxIdle	
minIdle	
initalSize	

1.6.7 C3P0连接池介绍

实现类ComPooledDataSource类。

1.6.8 C3P0连接池工具类编写

配置文件

```
<c3p0-config>

  <!--默认配置-->
    <default-config>

    <!-- initialPoolSize：初始化时获取三个连接，
        取值应在minPoolSize与maxPoolSize之间。 -->
        <property name="initialPoolSize">3</property>

    <!-- maxIdleTime：最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。-->
        <property name="maxIdleTime">60</property>

    <!-- maxPoolSize：连接池中保留的最大连接数 -->
        <property name="maxPoolSize">100</property>
    <!-- minPoolSize: 连接池中保留的最小连接数 -->
        <property name="minPoolSize">10</property>

    </default-config>

   <!--配置连接池mysql-->

    <named-config name="mysql">
        <property name="driverClass">com.mysql.cj.jdbc.Driver</property>
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/db1?
characterEncoding=UTF-8&amp;serverTimezone=UTC</property>
        <property name="user">root</property>
        <property name="password">Fzc1234!</property>
        <property name="initialPoolSize">10</property>
        <property name="maxIdleTime">30</property>
        <property name="maxPoolSize">100</property>
        <property name="minPoolSize">10</property>
    </named-config>
    <!--配置连接池2,可以配置多个-->

</c3p0-config>
```

工具类

```
public class C3P0Utils {

    public static ComboPooledDataSource dataSource = new
ComboPooledDataSource("mysql");

    public static Connection getConnection(){
        Connection connection = null;
        try {
            connection = dataSource.getConnection();
        } catch (SQLException exception) {
            exception.printStackTrace();
        }
        return connection;
    }

    public static void close(Connection con, Statement statement){
        if(con != null && statement != null){
            try {
                statement.close();
                con.close();
            } catch (SQLException exception) {
                exception.printStackTrace();
            }
        }
    }

    public static void close(Connection con, Statement statement, ResultSet
resultSet){
        if(con != null && statement != null && resultSet != null){
            try {
                resultSet.close();
                statement.close();
                con.close();
            } catch (SQLException exception) {
                exception.printStackTrace();
            }
        }
    }
}
```

1.6.10 Druid连接池介绍

Druid是阿里巴巴开发的连接池。它是目前最好的数据连接池。

1.6.11Druid连接池工具类编写

```
public class DruidUtils {

    public static DataSource dataSource;

    static {
        try {
            //创建属性集对象
            Properties p = new Properties();

            //Druid连接池不能够主动加载配置文件，需要指定文件
            InputStream resourceAsStream =
DruidUtils.class.getClassLoader().getResourceAsStream("druid.properties");

            //Properties对象的load方法可以从字节流读取配置信息
            p.load(resourceAsStream);

            //通过工厂类获取连接池对象
            dataSource = DruidDataSourceFactory.createDataSource(p);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection(){
        try {
            return dataSource.getConnection();
        } catch (SQLException exception) {
            exception.printStackTrace();
            return null;
        }
    }

    public static void close(Connection con, Statement statement){
        if(con != null && statement != null){
            try {
                statement.close();
                con.close();
            } catch (SQLException exception) {
                exception.printStackTrace();
```

```
            }
        }
    }

    public static void close(Connection con, Statement statement, ResultSet
resultSet){
        if(con != null && statement != null && resultSet != null){
            try {
                resultSet.close();
                statement.close();
                con.close();
            } catch (SQLException exception) {
                exception.printStackTrace();
            }
        }
    }
}
```

1.6.13 DBUtils工具类

DBUtils是Apache组织提供的一个对JDBC进行简单封装的开源工具库，使用它能够简化JDBC应用程序的开发，同时不会影响程序的性能。

核心功能

1. QueryRunner类中提供对sql语句操作的API
2. ResultSetHandler接口，用于定义select操作后，怎么封装结果集
3. DbUtils类是一个工具类定义了关闭资源与事务相关的方法

1.6.14 案例相关知识介绍

表与类之间的关系

可以将一张表看作一个类。每一列可以看作一个成员变量。一个对象对应一行数据。

JavaBean

JavaBean就是一个类，通常用于封装数据

1. 需要实现序列化接口Serializable
2. 提供私用字段
3. 提供getter和setter
4. 提供空参构造
5. 根据数据表对应建立的类

1.6.15 QueryRunner核心类的创建方式

语句	功能介绍
QueryRunner queryRunner1 = new QueryRunner();	手动
QueryRunner queryRunner2 = new QueryRunner(DruidUtils.getDataSource());	自动，提供连接池

1.6.16 QueryRunner类-实现插入操作

方法	功能介绍
update(Connection conn, String sql, Object... params)	Connection conn 数据库连接对象，自动创建可以不传，手动模式必须传；String sql 占位符的sql，使用？占位符；Object类型的可变参，用来设置占位符的参数。**可以实现增删改的操作**

方法	公共介绍
query(String sql, hander, Object... param)	自动模式创建
query(Connection con, String sql, hander, Object... param)	手动模式创建

1.6.18 ResultHandler结果集处理接口介绍

ResultSetHandler 实现类	说明
ArrayHandler	将结果集中的第一条记录封装到一个Object[]数组中，数组中的每一个元素就是这条记录中的每一个字段的值
ArrayListHandler	将结果集中的每一条记录都封装到一个Object[]数组中，将这些数组在封装到List集合中。
BeanHandler	将结果集中第一条记录封装到一个指定的javaBean中.
BeanListHandler	将结果集中每一条记录封装到指定的javaBean中，再将这些javaBean在封装到List集合中
ColumnListHandler	将结果集中指定的列的字段值，封装到一个List集合中
KeyedHandler	将结果集中每一条记录封装到Map<String,Object>,在将这个map集合做为另一个Map的value,另一个Map集合的key是指定的字段的值。
MapHandler	将结果集中第一条记录封装到了Map<String,Object>集合中，key就是字段名称，value就是字段值
MapListHandler	将结果集中每一条记录封装到了Map<String,Object>集合中，key就是字段名称，value就是字段值，在将这些Map封装到List集合中。
ScalarHandler	它是用于封装单个数据。例如 select count(*) from 表操作。

1.6.22 批处理介绍

一次操作多条sql语句。Statement和preparedStatement都支持批处理。

方法	功能介绍
void addBatch()	
int[] executeBatch()	

MySql预处理是默认的，所以需要加一个参数在url中

rewriteBatchedStatements=true

1.6.24 MySql元数据介绍&相关命令

除了表之外的数据都是元数据

1. 查询结果信息：update或delete语句 受影响的记录数
2. 数据库和数据表的信息
3. MySql服务信息

语句	功能介绍
show status;	参看数据库信息
select version();	MySql版本
show columns from 表名;	查询表的详细信息
show index from 表名;	显示数据表的详细索引信息
show databases;	列出所有数据库
show tables;	显示当前数据库的所有表
select database();	获取当前数据库名

1.6.25 JDBC获取元数据常用类介绍

元数据类	功能介绍
DatabaseMetaData	描述数据库的元数据对象
ResultSetMetaData	描述结果集的元数据对象

通过getMetaData()方法获取

1. 使用Connection对象调用，获取的是DatabaseMetaData
2. 使用PreparedStatement对象调用，获取的是ResultSetMetaData

1.6.26 JDBC获取数据库元数据信息

方法	功能介绍
getUrl()	
getUsername()	
getDatabaseProductName()	
getDatabaseProductVersion()	
getDriverName()	
isReadOnly()	

1.6.27 JDBC获取结果集元数据信息

方法	功能
getColumnCount()	
getColumnName(int i)	参数是从1开始的
getColumnTypeName(int i)	参数是从1开始的

1.7 XML

1.7.1 XML基本介绍

XML是可扩展标记语言(Extensible Markup Language)

1. 语法严格
2. 可扩展，标签都是自己定义的

XML能做什么：

1. 可以存储数据
2. 作为配置文件使用
3. 使用XML在网络中传输数据

1.7.2 XML的语法格式

```
<!-- XML的注释
    1.必须有版本申明
    2.文档申明必须写在第一行
    3.XML的元素标签：
        1.不能使用空格 或冒号
        2.区分大小写
    4.有且只有一个根元素
    5.元素体可以是文本，也可以是一个标签
    6.空元素，没有结束标签
 -->
```

1.7.3 自定义xml描述数据

1.7.4 XML约束介绍

在XML技术里，可以编写一个文档来约束一个XML文档的书写规范。

常见的xml约束：

1. DTD
2. Schema

程序员编写XML；框架解析XML并编写约束文档文件。程序员根据约束文档编写XML来适应不同的框架。

1.7.5 DTD约束介绍和编写

规定XML文档中元素的名称，子元素的名称及顺序，元素的属性等。

1.7.6 引入DTD约束

语句	功能介绍

1.7.7 Schema约束介绍和编写

以xsd结尾

1.7.8 引入Schema约束

1.7.9 XML解析方式介绍

1. DOM方式：将整个XML读取到内存中生成一个document对象

   优点：元素与元素之间有结构关系，可以进行CRUD

   缺点：占用的内存太多，容易溢出
2. SAX方式：一边扫描一边解析，速度快

   优点：占用内存少，速度快

   缺点：只能进行读操作

1.7.10 常见的XML解析器

1. JAXP：支持DOM和SAX
2. DOM4J：支持DOM和SAX
3. Jsoup：
4. PULL：

1.7.11 DOM4JAPI介绍

核心类SaxReader加载XML文档到内存，获得Document对象。

方法	功能介绍
read()	加载xml，返回Document对象。通过Document对象获取到根元素

Document对象

方法	功能介绍
getRootElement()	获取根元素
selectSingleNode()	

Element对象

方法	功能介绍
elements(...)	获取指定名称的所有子元素
element(...)	获取指定名称的第一个子元素
getName()	获取当前元素的元素名
attribtueValue(...)	获取指定属性名的属性值
elementText(...)	获取指定名称子元素的文本值
getText()	获取当前元素的文本内容

1.7.15 XPath介绍

XPath是一门在XML文档中查找信息的语言。

1.7.16 ～ 1.7.21 待看