mysql多表

学习目标

1. 能够使用内连接进行多表查询
2. 能够使用左外连接和右外连接进行多表查询
3. 能够理解多表查询的规律
4. 能够使用子查询
5. 能够理解事务的概念
6. 能够说出事务的原理
7. 能够在MySQL中使用事务
8. 能够理解脏读,不可重复读,幻读的概念及解决办法

第1章 表关系

目标

创建多表，并往多表中添加数据

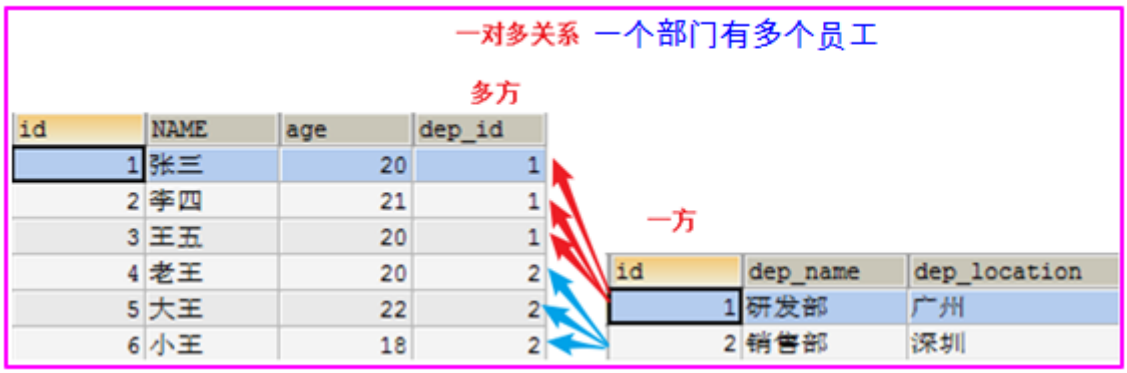
1.1 表关系的概念

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！分成三种：

1. 一对一
2. 一对多
3. 多对多

1.2 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,指向主表(一方)的主键.我们把这个字段称之为外键.



1.3 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

1.4 一对一 (了解)

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：

- 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

1.5 外键约束

1.4.1 什么是外键约束

一张表中的某个字段引用另一个表的主键 主表： 约束别人 副表/从表： 使用别人的数据，被别人约束

一张表中的某个字段引用另一个表的主键

employee员工表　　外键

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

副表/从表： 使用别人的数据，被别人约束

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

1.4.2 创建外键

1. **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)` **关键字解释**： CONSTRAINT -- 约束关键字 FOREIGN KEY(外键字段名) -- 某个字段作为外键 REFERENCES -- 主表名(主键字段名) 表示参照主表中的某个字段
2. **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：以"新建表时添加外键"演示

```
-- 先创建部门表
CREATE TABLE department (
    id INT PRIMARY KEY AUTO_INCREMENT,
    dep_name VARCHAR(20),
    dep_location VARCHAR(20)
);

-- 添加2个部门
INSERT INTO department (dep_name, dep_location) VALUES ('研发部', '广州'), ('销售部', '深圳');

-- 然后创建员工表,添加外键约束
CREATE TABLE employee (
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(20),
    age INT,
    dep_id INT,
    -- 添加一个外键
    -- 外键取名公司要求,一般fk结尾
```

```
    CONSTRAINT emp_depid_ref_dep_id_fk FOREIGN KEY(dep_id) REFERENCES
department(id)
);
```

- 正常添加数据

```
INSERT INTO employee (NAME, age, dep_id) VALUES
('张三', 20, 1),
('李四', 21, 1),
('王五', 20, 1),
('老王', 20, 2),
('大王', 22, 2),
('小王', 18, 2);
```

- 部门错误的数据添加失败

```
INSERT INTO employee (NAME, age, dep_id) VALUES ('二王', 20, 5);
```

1.4.4 删除外键(了解)

`ALTER TABLE 从表 drop foreign key 外键名称;`

具体操作:

- 删除`employee`表的`emp_depid_ref_dep_id_fk`外键

```
ALTER TABLE employee DROP FOREIGN KEY emp_depid_ref_dep_id_fk;
```

- 在`employee`表情存在况下添加外键

```
ALTER TABLE employee ADD CONSTRAINT emp_depid_ref_dep_id_fk FOREIGN KEY(dep_id)
REFERENCES department(id);
```

1.5 多表练习

一对多关系练习

以下案例是我们JavaWeb课程最后的小项目.我们拿出其中一部分需求,根据需求来设计数据库表之间的关系

一个旅游线路分类中有多条旅游线路，一条旅游线路属于某一个分类 . 旅游线路表是多表 ,可以在多表上添加一个外键来指向分类表中的主键.

具体操作:

- 创建旅游线路分类表

```
CREATE TABLE tab_category (
    cid INT PRIMARY KEY AUTO_INCREMENT, -- 旅游线路分类主键
    cname VARCHAR(100) NOT NULL UNIQUE -- 旅游线路分类名称
);
```

- 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');
```

- 创建旅游线路表

```
CREATE TABLE tab_route (
    rid INT PRIMARY KEY AUTO_INCREMENT, -- 旅游线路主键
    rname VARCHAR(100) NOT NULL UNIQUE, -- 旅游线路名称
    price DOUBLE NOT NULL,  -- 价格
    cid INT NOT NULL, -- 所属分类
    CONSTRAINT ro_cid_ref_cate_id FOREIGN KEY(cid) REFERENCES tab_category(cid)
);
```

- 添加旅游线路测试数据(自己添加)

多对多关系练习

一个用户可以收藏多个线路，一个线路可以被多个用户收藏 ,所以用户和线路之间是多对多的关系.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

具体操作：

- 创建用户表

```
CREATE TABLE tab_user (
   uid INT PRIMARY KEY AUTO_INCREMENT, -- 用户id
   username VARCHAR(100) NOT NULL UNIQUE -- 用户名
);
```

- 添加用户数据

```
INSERT INTO tab_user VALUES
(NULL, '老王'),
(NULL, '小王');
```

- 创建收藏表

```
CREATE TABLE tab_favorite (
   fid INT PRIMARY KEY AUTO_INCREMENT, -- 收藏主键
   rid INT NOT NULL, -- 旅游线路id
   DATE DATE NOT NULL, -- 收藏时间
   uid INT NOT NULL -- 用户id
);
```

> 注意：可以自己尝试着给收藏表添加外键约束

- 增加收藏表测试数据(自己添加)

小结

多表指的其实就是表中的数据和其他表中的数据有一定的关系，而这关系不是由数据库决定的，是我们添加数据的时候指定了某一列的数据指向了另一个表，通过这种方式我们可以操作多表

> 创建外键 `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);` 删除外键 `ALTER TABLE 表名 DROP FOREIGN KEY 外键名称;`

第2章 多表查询

目标

理解多表查询操作

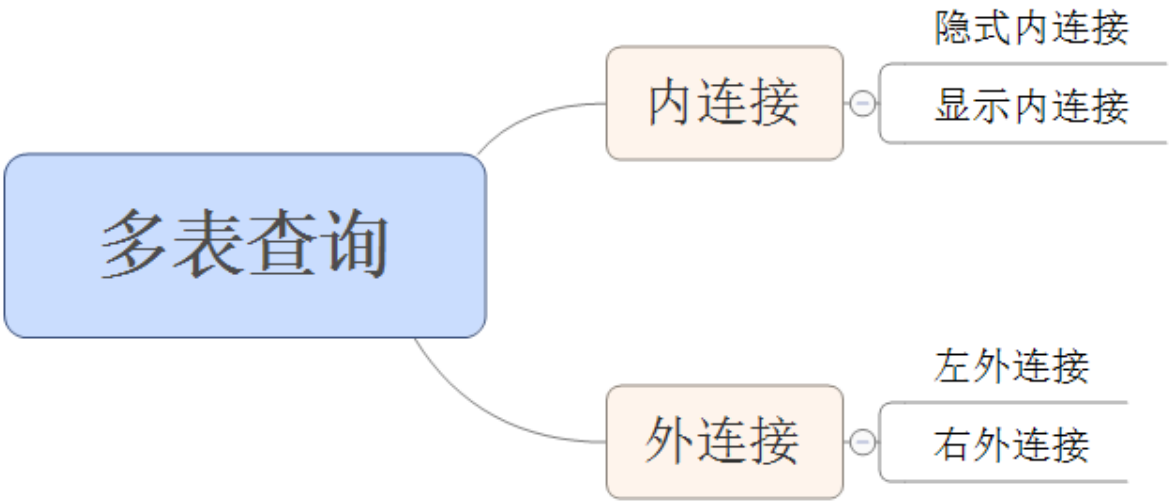
2.1 什么是多表查询

同时查询多张表获取到需要的数据 比如：我们想查询到开发部有多少人，需要将部门表和员工表同时进行查询

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

id	NAME
1	开发部
2	市场部
3	财务部

多表查询的分类:



准备数据:

```
-- 创建部门表
CREATE TABLE dept (
  id INT PRIMARY KEY AUTO_INCREMENT,
  NAME VARCHAR(20)
);

INSERT INTO dept (NAME) VALUES ('开发部'),('市场部'),('财务部');

-- 创建员工表
CREATE TABLE emp (
  id INT PRIMARY KEY AUTO_INCREMENT,
  NAME VARCHAR(10),
  gender CHAR(1),   -- 性别
  salary DOUBLE,   -- 工资
  join_date DATE,  -- 入职日期
  dept_id INT
);

INSERT INTO emp(NAME,gender,salary,join_date,dept_id) VALUES('孙悟
空','男',7200,'2013-02-24',1);
INSERT INTO emp(NAME,gender,salary,join_date,dept_id) VALUES('猪八
戒','男',3600,'2010-12-02',2);
INSERT INTO emp(NAME,gender,salary,join_date,dept_id) VALUES('唐
僧','男',9000,'2008-08-08',2);
INSERT INTO emp(NAME,gender,salary,join_date,dept_id) VALUES('白骨
精','女',5000,'2015-10-07',3);
INSERT INTO emp(NAME,gender,salary,join_date,dept_id) VALUES('蜘蛛
精','女',4500,'2011-03-14',1);
```

2.2 笛卡尔积现象

2.2.1 什么是笛卡尔积现象

多表查询时左表的每条数据和右表的每条数据组合，这种效果称为笛卡尔积

需求：查询每个部门有哪些人

具体操作：

```
SELECT * FROM dept, emp;
```

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	1	孙悟空	男	7200	2013-02-24	1
2	市场部	1	孙悟空	男	7200	2013-02-24	1
3	财务部	1	孙悟空	男	7200	2013-02-24	1
1	开发部	2	猪八戒	男	3600	2010-12-02	2
2	市场部	2	猪八戒	男	3600	2010-12-02	2
3	财务部	2	猪八戒	男	3600	2010-12-02	2
1	开发部	3	唐僧	男	9000	2008-08-08	2
2	市场部	3	唐僧	男	9000	2008-08-08	2
3	财务部	3	唐僧	男	9000	2008-08-08	2
1	开发部	4	白骨精	女	5000	2015-10-07	3
2	市场部	4	白骨精	女	5000	2015-10-07	3
3	财务部	4	白骨精	女	5000	2015-10-07	3
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
2	市场部	5	蜘蛛精	女	4500	2011-03-14	1
3	财务部	5	蜘蛛精	女	4500	2011-03-14	1

以上数据其实是左表的每条数据和右表的每条数据组合。左表有3条，右表有5条，最终组合后3*5=15条数据。

左表的每条数据和右表的每条数据组合，这种效果称为笛卡尔积

id	NAME
1	开发部
2	市场部
3	财务部

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

左表的每条数据和右表的每条数据组合，这种效果称为笛卡尔积

2.2.2 如何清除笛卡尔积现象的影响

我们发现不是所有的数据组合都是有用的，只有员工表.dept_id = 部门表.id 的数据才是有用的。所以需要通过条件过滤掉没用的数据。

id	NAME
1	开发部
2	市场部
3	财务部

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

左表的每条数据和右表的每条数据组合，这种效果称为笛卡尔积

我们发现不是所有的数据组合都是有用的，只有员工表中的dept_id = 部门表中id的数据才是有用的，所以需要通过条件过滤掉没用的数据

```
SELECT * FROM dept, emp WHERE emp.`dept_id`=dept.`id`;
```

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
1	开发部	1	孙悟空	男	7200	2013-02-24	1
2	市场部	3	唐僧	男	9000	2008-08-08	2
2	市场部	2	猪八戒	男	3600	2010-12-02	2
3	财务部	4	白骨精	女	5000	2015-10-07	3

2.3 内连接

用左边表的记录去匹配右边表的记录，如果符合条件的则显示

2.3.1 隐式内连接

隐式内连接：看不到 `JOIN` 关键字，条件使用 `WHERE` 指定 `SELECT 字段名 FROM 左表，右表 WHERE 条件；`

2.3.2 显示内连接

显示内连接：使用 `INNER JOIN ... ON` 语句, 可以省略 `INNER` `SELECT 字段名 FROM 左表 INNER JOIN 右表 ON 条件；`

具体操作：

- 查询唐僧的信息，显示员工id，姓名，性别，工资和所在的部门名称，我们发现需要联合2张表同时才能查询出需要的数据，我们使用内连接

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

1. 确定查询哪些表

```
SELECT * FROM dept INNER JOIN emp;
```

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	1	孙悟空	男	7200	2013-02-24	1
2	市场部	1	孙悟空	男	7200	2013-02-24	1
3	财务部	1	孙悟空	男	7200	2013-02-24	1
1	开发部	2	猪八戒	男	3600	2010-12-02	2
2	市场部	2	猪八戒	男	3600	2010-12-02	2
3	财务部	2	猪八戒	男	3600	2010-12-02	2
1	开发部	3	唐僧	男	9000	2008-08-08	2
2	市场部	3	唐僧	男	9000	2008-08-08	2
3	财务部	3	唐僧	男	9000	2008-08-08	2
1	开发部	4	白骨精	女	5000	2015-10-07	3
2	市场部	4	白骨精	女	5000	2015-10-07	3
3	财务部	4	白骨精	女	5000	2015-10-07	3
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
2	市场部	5	蜘蛛精	女	4500	2011-03-14	1
3	财务部	5	蜘蛛精	女	4500	2011-03-14	1

1. 确定表连接条件，员工表.dept_id = 部门表.id 的数据才是有效的

```
SELECT * FROM dept INNER JOIN emp ON emp.`dept_id`=dept.`id`;
```

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
1	开发部	1	孙悟空	男	7200	2013-02-24	1
2	市场部	3	唐僧	男	9000	2008-08-08	2
2	市场部	2	猪八戒	男	3600	2010-12-02	2
3	财务部	4	白骨精	女	5000	2015-10-07	3

1. 确定表连接条件，我们查询的是唐僧的信息，部门表.name='唐僧'

```
SELECT * FROM dept INNER JOIN emp ON emp.`dept_id`=dept.`id` AND emp.`NAME`='唐僧';
```

id	NAME	id	NAME	gender	salary	join_date	dept_id
2	市场部	3	唐僧	男	9000	2008-08-08	2

1. 确定查询字段，查询唐僧的信息，显示员工id，姓名，性别，工资和所在的部门名称

```
SELECT emp.`id`, emp.`NAME`, emp.`gender`, emp.`salary`, dept.`NAME` FROM dept INNER JOIN emp ON emp.`dept_id`=dept.`id` AND emp.`NAME`='唐僧';
```

id	NAME	gender	salary	NAME
3	唐僧	男	9000	市场部

1. 我们发现写表名有点长，可以给表取别名，显示的字段名也使用别名

```
SELECT e.`id` 员工编号, e.`NAME` 员工姓名, e.`gender` 性别, e.`salary` 工资, d.`NAME` 部门名称 FROM dept d INNER JOIN emp e ON e.`dept_id`=d.`id` AND e.`NAME`='唐僧';
```

员工编号	员工姓名	性别	工资	部门名称
3	唐僧	男	9000	市场部

总结内连接查询步骤：

1. 确定查询哪些表
2. 确定表连接条件
3. 确定查询字段

2.4 左外连接

左外连接：使用`LEFT OUTER JOIN ... ON`，`OUTER`可以省略`SELECT 字段名 FROM 左表 LEFT OUTER JOIN 右表 ON 条件;`用左边表的记录去匹配右边表的记录，如果符合条件的则显示；否则，显示NULL 可以理解为：在内连接的基础上保证左表的数据全部显示

具体操作：

- 在部门表中增加一个销售部

```
INSERT INTO dept (NAME) VALUES ('销售部');
```

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

- 使用内连接查询

```
SELECT * FROM dept INNER JOIN emp ON emp.`dept_id`=dept.`id`;
```

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	1	孙悟空	男	7200	2013-02-24	1
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
2	市场部	2	猪八戒	男	3600	2010-12-02	2
2	市场部	3	唐僧	男	9000	2008-08-08	2
3	财务部	4	白骨精	女	5000	2015-10-07	3

内连接：用左边表的记录去匹配右边表的记录，如果符合条件的则显示

- 使用左外连接查询

```
SELECT * FROM dept LEFT OUTER JOIN emp ON emp.`dept_id`=dept.`id`;
```

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	1	孙悟空	男	7200	2013-02-24	1
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
2	市场部	2	猪八戒	男	3600	2010-12-02	2
2	市场部	3	唐僧	男	9000	2008-08-08	2
3	财务部	4	白骨精	女	5000	2015-10-07	3
4	销售部	*(NULL)*	*(NULL)*	*(NULL)*	*(NULL)*	*(NULL)*	*(NULL)*

用左边表的记录去匹配右边表的记录，如果符合条件的则显示；否则，显示NULL
可以理解为：在内连接的基础上保证左表的数据全部显示

2.5 右外连接

右外连接：使用`RIGHT OUTER JOIN ... ON`，`OUTER`可以省略`SELECT 字段名 FROM 左表 RIGHT OUTER JOIN 右表 ON 条件;`用右边表的记录去匹配左边表的记录，如果符合条件的则显示；否则，显示NULL 可以理解为：在内连接的基础上保证右表的数据全部显示

具体操作：

- 在员工表中增加一个员工 INSERT INTO emp(NAME,gender,salary,join_date,dept_id) VALUES('沙僧','男',6666,'2013-02-24',NULL);

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1
6	沙僧	男	6666	2013-02-24	(NULL)

- 使用内连接查询 SELECT * FROM dept INNER JOIN emp ON emp.`dept_id`=dept.`id`;

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	1	孙悟空	男	7200	2013-02-24	1
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
2	市场部	2	猪八戒	男	3600	2010-12-02	2
2	市场部	3	唐僧	男	9000	2008-08-08	2
3	财务部	4	白骨精	女	5000	2015-10-07	3

内连接：用左边表的记录去匹配右边表的记录，如果符合条件的则显示

- 使用右外连接查询 SELECT * FROM dept RIGHT OUTER JOIN emp ON emp.`dept_id`=dept.`id`;

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	1	孙悟空	男	7200	2013-02-24	1
2	市场部	2	猪八戒	男	3600	2010-12-02	2
2	市场部	3	唐僧	男	9000	2008-08-08	2
3	财务部	4	白骨精	女	5000	2015-10-07	3
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
(NULL)	(NULL)	6	沙僧	男	6666	2013-02-24	(NULL)

用右边表的记录去匹配左边表的记录，如果符合条件的则显示；否则，显示NULL
可以理解为：在内连接的基础上保证右表的数据全部显示

第3章 子查询

一条SELECT语句结果作为另一条SELECT语法一部分（作为查询条件或查询结果或表） `SELECT 查询字段 FROM 表 WHERE 查询条件;` `SELECT * FROM employee WHERE salary=(SELECT MAX(salary) FROM employee);`

```
SELECT * FROM employee WHERE salary=(SELECT MAX(salary) FROM employee);
```

子查询

子查询需要放在（）中

子查询结果的三种情况：

1. 子查询的结果是一个值的时候

max(salary)
9000

2. 子查询结果是单列多行的时候

dept_id
1
2

3. 子查询的结果是多行多列的时候

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1
6	沙僧	男	6666	2013-02-24	*(NULL)*

> 说明： 子查询结果只要是`单列`，多数情况下会在`WHERE`后面作为`条件` 子查询结果只要是`多列`，肯定在`FROM`后面作为`表`

3.1 子查询的结果是一个值

子查询结果是`单列`，多数情况下会在`WHERE`后面作为`条件` `SELECT 查询字段 FROM 表 WHERE 字段=（子查询）;`

1. **查询工资最高的员工是谁?**
   - 查询最高工资是多少

```
SELECT MAX(salary) FROM emp;
```

MAX(salary)
9000

   - 根据最高工资到员工表查询到对应的员工信息

```
SELECT * FROM emp WHERE salary=(SELECT MAX(salary) FROM emp);
```

id	NAME	gender	salary	join_date	dept_id
3	唐僧	男	9000	2008-08-08	2

2. **查询工资小于平均工资的员工有哪些?**
   - 查询平均工资是多少

```
SELECT AVG(salary) FROM emp;
```

avg(salary)
5994.333333333333

   - 到员工表查询小于平均的员工信息

```
SELECT * FROM emp WHERE salary < (SELECT AVG(salary) FROM emp);
```

id	NAME	gender	salary	join_date	dept_id
2	猪八戒	男	3600	2010-12-02	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

3.2 子查询结果是单列多行

子查询结果是`单列`，多数情况下会在`WHERE`后面作为`条件` 子查询结果是单例多行，结果集类似于一个数组，父查询使用`IN`或`NOT IN`等运算符进行条件筛选 `SELECT 查询字段 FROM 表 WHERE 字段 IN（子查询）;`

1. **查询工资大于5000的员工，来自于哪些部门的名字**

- 先查询大于5000的员工所在的部门id

```
  SELECT dept_id FROM emp WHERE salary > 5000;
```

SELECT dept_id FROM emp WHERE salary > 5000;

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1
6	沙僧	男	6666	2013-02-24	(NULL)

dept_id
1
2
(NULL)

- 再查询在这些部门id中部门的名字

```
  SELECT dept.name FROM dept WHERE dept.id IN (SELECT dept_id FROM emp WHERE
salary > 5000);
```

SELECT dept_id FROM emp WHERE salary > 5000;

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1
6	沙僧	男	6666	2013-02-24	(NULL)

dept_id
1
2
(NULL)

SELECT dept.name FROM dept WHERE dept.id IN (SELECT dept_id FROM emp WHERE salary > 5000);

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

name
开发部
市场部

2. **查询开发部与财务部所有的员工信息**
   - 先查询开发部与财务部的id

```
SELECT id FROM dept WHERE NAME IN('开发部','财务部');
```

SELECT id FROM dept WHERE NAME IN('开发部','财务部');

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

id
1
3

   - 再查询在这些部门id中有哪些员工

```
SELECT * FROM emp WHERE dept_id IN (SELECT id FROM dept WHERE NAME IN('开发
部','财务部'));
```

SELECT id FROM dept WHERE NAME IN('开发部','财务部');

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

id
1
3

SELECT * FROM emp WHERE dept_id IN
(SELECT id FROM dept WHERE NAME IN('开发部','财务部'));

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
2	猪八戒	男	3600	2010-12-02	2
3	唐僧	男	9000	2008-08-08	2
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1
6	沙僧	男	6666	2013-02-24	(NULL)

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1

3.3 子查询的结果是多行多列

子查询结果是`多列`，多数情况下放在`FROM`后面作为`表` `SELECT 查询字段 FROM （子查询） 表别名 WHERE 条件;` 子查询作为表需要取别名，否则这张表没用名称无法访问表中的字段

- **查询出2011年以后入职的员工信息，包括部门名称**
  - 在员工表中查询2011-1-1以后入职的员工

```
SELECT * FROM emp WHERE join_date > '2011-1-1';
```

在员工表中查询2011-1-1以后入职的员工
SELECT * FROM emp WHERE join_date > '2011-1-1';

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1
6	沙僧	男	6666	2013-02-24	(NULL)

  - 查询所有的部门信息，与上面的虚拟表中的信息组合，找出所有部门id等于的dept_id

```
SELECT * FROM dept d, (SELECT * FROM emp WHERE join_date > '2011-1-1') e
WHERE e.dept_id = d.id;
```

查询所有的部门信息，与上面的虚拟表中的信息组合，找出所有部门id等于的dept_id
SELECT * FROM dept d, (SELECT * FROM emp WHERE join_date > '2011-1-1') e WHERE e.dept_id = d.id;

id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

id	NAME	gender	salary	join_date	dept_id
1	孙悟空	男	7200	2013-02-24	1
4	白骨精	女	5000	2015-10-07	3
5	蜘蛛精	女	4500	2011-03-14	1
6	沙僧	男	6666	2013-02-24	(NULL)

id	NAME	id	NAME	gender	salary	join_date	dept_id
1	开发部	1	孙悟空	男	7200	2013-02-24	1
1	开发部	5	蜘蛛精	女	4500	2011-03-14	1
3	财务部	4	白骨精	女	5000	2015-10-07	3

使用表连接：

```
SELECT d.*, e.* FROM dept d INNER JOIN emp e ON d.id = e.dept_id WHERE
e.join_date > '2011-1-1';
```

3.4 多表查询总结

> - 笛卡尔积查询： select * from A , B 表示两个表数据的乘积
> - 内连接

    - 显示内连接
        - select * from A [inner] join B on 表关联条件
    - 隐式内连接
        - select * from A , B where 表关联条件
- 外连接
    - 左外连接
        - select * from A left [outer] join B on 表关联条件
    - 右外连接
        - select * from A right [outer] join B on 表关联条件
- 子查询结果是`单列`，多数情况下放在`WHERE`后面作为`条件` select 查询字段 from 表 where 字段=（子查询）；
- 子查询结果是`多列`，多数情况下放在`FROM`后面作为`表` select 查询字段 from （子查询）表别名 where 条件;

第4章 多表查询案例

我们在公司开发中，根据不同的业务需求往往需要通过2张及以上的表中去查询需要的数据。所以我们有必要学习2张及以上的表的查询。其实不管是几张表的查询，都是有规律可循的。

4.1 准备数据

```
-- 部门表
CREATE TABLE dept (
  id INT PRIMARY KEY PRIMARY KEY, -- 部门id
  dname VARCHAR(50), -- 部门名称
  loc VARCHAR(50) -- 部门位置
);

-- 添加4个部门
INSERT INTO dept(id,dname,loc) VALUES
(10,'教研部','北京'),
(20,'学工部','上海'),
(30,'销售部','广州'),
(40,'财务部','深圳');

-- 职务表，职务名称，职务描述
CREATE TABLE job (
  id INT PRIMARY KEY,
  jname VARCHAR(20),
  description VARCHAR(50)
);

-- 添加4个职务
INSERT INTO job (id, jname, description) VALUES
(1, '董事长', '管理整个公司，接单'),
(2, '经理', '管理部门员工'),
(3, '销售员', '向客人推销产品'),
(4, '文员', '使用办公软件');

-- 员工表
CREATE TABLE emp (
  id INT PRIMARY KEY, -- 员工id
```

```
  ename VARCHAR(50), -- 员工姓名
  job_id INT, -- 职务id
  mgr INT , -- 上级领导
  joindate DATE, -- 入职日期
  salary DECIMAL(7,2), -- 工资
  bonus DECIMAL(7,2), -- 奖金
  dept_id INT, -- 所在部门编号
  CONSTRAINT emp_jobid_ref_job_id_fk FOREIGN KEY (job_id) REFERENCES job (id),
  CONSTRAINT emp_deptid_ref_dept_id_fk FOREIGN KEY (dept_id) REFERENCES dept
(id)
);

-- 添加员工
INSERT INTO emp(id,ename,job_id,mgr,joindate,salary,bonus,dept_id) VALUES
(1001,'孙悟空',4,1004,'2000-12-17','8000.00',NULL,20),
(1002,'卢俊义',3,1006,'2001-02-20','16000.00','3000.00',30),
(1003,'林冲',3,1006,'2001-02-22','12500.00','5000.00',30),
(1004,'唐僧',2,1009,'2001-04-02','29750.00',NULL,20),
(1005,'李逵',4,1006,'2001-09-28','12500.00','14000.00',30),
(1006,'宋江',2,1009,'2001-05-01','28500.00',NULL,30),
(1007,'刘备',2,1009,'2001-09-01','24500.00',NULL,10),
(1008,'猪八戒',4,1004,'2007-04-19','30000.00',NULL,20),
(1009,'罗贯中',1,NULL,'2001-11-17','50000.00',NULL,10),
(1010,'吴用',3,1006,'2001-09-08','15000.00','0.00',30),
(1011,'沙僧',4,1004,'2007-05-23','11000.00',NULL,20),
(1012,'李逵',4,1006,'2001-12-03','9500.00',NULL,30),
(1013,'小白龙',4,1004,'2001-12-03','30000.00',NULL,20),
(1014,'关羽',4,1007,'2002-01-23','13000.00',NULL,10);

-- 工资等级表
CREATE TABLE salarygrade (
  grade INT PRIMARY KEY,
  losalary INT,
  hisalary INT
);

-- 添加5个工资等级
INSERT INTO salarygrade(grade,losalary,hisalary) VALUES
(1,7000,12000),
(2,12010,14000),
(3,14010,20000),
(4,20010,30000),
(5,30010,99990);
```

分析4张表的关系：通过4张表可以查出一个员工的所有信息

emp 员工表　　外键

id	ename	job_id	mgr	joindate	salary	bonus	dept_id
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20
1005	李逵	4	1006	2001-09-28	12500.00	14000.00	30
1006	宋江	2	1009	2001-05-01	28500.00	(NULL)	30
1007	刘备	2	1009	2001-09-01	24500.00	(NULL)	10
1008	猪八戒	4	1004	2007-04-19	30000.00	(NULL)	20
1009	罗贯中	1	(NULL)	2001-11-17	50000.00	(NULL)	10
1010	吴用	3	1006	2001-09-08	15000.00	0.00	30
1011	沙僧	4	1004	2007-05-23	11000.00	(NULL)	20
1012	李逵	4	1006	2001-12-03	9500.00	(NULL)	30
1013	小白龙	4	1004	2001-12-03	30000.00	(NULL)	20
1014	关羽	4	1007	2002-01-23	13000.00	(NULL)	10

dept 部门表

id 主键	dname	loc
10	教研部	北京
20	学工部	上海
30	销售部	广州
40	财务部	深圳

job 职务表

id 主键	jname	description
1	董事长	管理整个公司，接单
2	经理	管理部门员工
3	销售员	向客人推销产品
4	文员	使用办公软件

salarygrade 工资等级表

grade	losalary	hisalary
1	7000	12000
2	12010	14000
3	14010	20000
4	20010	30000
5	30010	99990

我们通过4张表的信息组合起来可以看到孙悟空的完整信息：
id：1001　姓名：孙悟空　职务：文员　入职日期：2000-12-17　工资：8000　工资等级：1级　奖金：0　部门：学工部

4.2 练习

4.2.1 练习1

查询所有员工信息。显示员工编号，员工姓名，工资，职务名称，职务描述

具体操作： 1.确定要查询哪些表：emp e, job j

```
SELECT * FROM emp e INNER JOIN job j;
```

id	ename	job_id	mgr	joindate	salary	bonus	dept_id	id	jname	description
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20	1	董事长	管理整个公司，接单
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20	2	经理	管理部门员工
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20	3	销售员	向客人推销产品
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20	4	文员	使用办公软件
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30	1	董事长	管理整个公司，接单
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30	2	经理	管理部门员工
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30	3	销售员	向客人推销产品
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30	4	文员	使用办公软件
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30	1	董事长	管理整个公司，接单
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30	2	经理	管理部门员工
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30	3	销售员	向客人推销产品
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30	4	文员	使用办公软件
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20	1	董事长	管理整个公司，接单
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20	2	经理	管理部门员工
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20	3	销售员	向客人推销产品

LECT * FROM emp e INNER JOIN job j LIMIT 0, 1000

执行：0.003 sec　总数：0.005 sec　56 行　Ln 96, Col 55　连接：1

总共出现了56条数据，左表14x右表4=56，笛卡尔积，其中很多是没用的数据

emp 员工表　　外键　　14条记录

id	ename	job_id	mgr	joindate	salary	bonus	dept_id
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20
1005	李逵	4	1006	2001-09-28	12500.00	14000.00	30
1006	宋江	2	1009	2001-05-01	28500.00	(NULL)	30
1007	刘备	2	1009	2001-09-01	24500.00	(NULL)	10
1008	猪八戒	4	1004	2007-04-19	30000.00	(NULL)	20
1009	罗贯中	1	(NULL)	2001-11-17	50000.00	(NULL)	10
1010	吴用	3	1006	2001-09-08	15000.00	0.00	30
1011	沙僧	4	1004	2007-05-23	11000.00	(NULL)	20
1012	李逵	4	1006	2001-12-03	9500.00	(NULL)	30
1013	小白龙	4	1004	2001-12-03	30000.00	(NULL)	20
1014	关羽	4	1007	2002-01-23	13000.00	(NULL)	10

job 职务表　4条记录

id 主键	jname	description
1	董事长	管理整个公司，接单
2	经理	管理部门员工
3	销售员	向客人推销产品
4	文员	使用办公软件

2.确定表连接条件： e.job_id=j.id

```
SELECT * FROM emp e INNER JOIN job j ON e.job_id=j.id;
```

这2张表之间的关系，是通过emp.job_id 和job.id进行关联的

只有emp.job_id=job.id的数据才是有用的

emp 员工表

id	ename	job_id	mgr	joindate	salary	bonus	dept_id
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20
1005	李逵	4	1006	2001-09-28	12500.00	14000.00	30
1006	宋江	2	1009	2001-05-01	28500.00	(NULL)	30
1007	刘备	2	1009	2001-09-01	24500.00	(NULL)	10
1008	猪八戒	4	1004	2007-04-19	30000.00	(NULL)	20
1009	罗贯中	1	(NULL)	2001-11-17	50000.00	(NULL)	10
1010	吴用	3	1006	2001-09-08	15000.00	0.00	30
1011	沙僧	4	1004	2007-05-23	11000.00	(NULL)	20
1012	李逵	4	1006	2001-12-03	9500.00	(NULL)	30
1013	小白龙	4	1004	2001-12-03	30000.00	(NULL)	20
1014	关羽	4	1007	2002-01-23	13000.00	(NULL)	10

job 职务表

id	jname	description
1	董事长	管理整个公司，接单
2	经理	管理部门员工
3	销售员	向客人推销产品
4	文员	使用办公软件

id	ename	job_id	mgr	joindate	salary	bonus	dept_id	id	jname	description
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20	4	文员	使用办公软件
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30	3	销售员	向客人推销产品
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30	3	销售员	向客人推销产品
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20	2	经理	管理部门员工
1005	李逵	4	1006	2001-09-28	12500.00	14000.00	30	4	文员	使用办公软件
1006	宋江	2	1009	2001-05-01	28500.00	(NULL)	30	2	经理	管理部门员工
1007	刘备	2	1009	2001-09-01	24500.00	(NULL)	10	2	经理	管理部门员工
1008	猪八戒	4	1004	2007-04-19	30000.00	(NULL)	20	4	文员	使用办公软件
1009	罗贯中	1	(NULL)	2001-11-17	50000.00	(NULL)	10	1	董事长	管理整个公司，接单
1010	吴用	3	1006	2001-09-08	15000.00	0.00	30	3	销售员	向客人推销产品
1011	沙僧	4	1004	2007-05-23	11000.00	(NULL)	20	4	文员	使用办公软件
1012	李逵	4	1006	2001-12-03	9500.00	(NULL)	30	4	文员	使用办公软件
1013	小白龙	4	1004	2001-12-03	30000.00	(NULL)	20	4	文员	使用办公软件
1014	关羽	4	1007	2002-01-23	13000.00	(NULL)	10	4	文员	使用办公软件

3.确定查询字段：员工编号，员工姓名，工资，职务名称，职务描述

```
SELECT e.`id`, e.`ename`, e.`salary`, j.`jname`, j.`description` FROM emp e INNER JOIN job j ON e.job_id=j.id;
```

id	ename	salary	jname	description
1009	罗贯中	50000.00	董事长	管理整个公司，接单
1004	唐僧	29750.00	经理	管理部门员工
1006	宋江	28500.00	经理	管理部门员工
1007	刘备	24500.00	经理	管理部门员工
1002	卢俊义	16000.00	销售员	向客人推销产品
1003	林冲	12500.00	销售员	向客人推销产品
1010	吴用	15000.00	销售员	向客人推销产品
1001	孙悟空	8000.00	文员	使用办公软件
1005	李逵	12500.00	文员	使用办公软件
1008	猪八戒	30000.00	文员	使用办公软件
1011	沙僧	11000.00	文员	使用办公软件
1012	李逵	9500.00	文员	使用办公软件
1013	小白龙	30000.00	文员	使用办公软件
1014	关羽	13000.00	文员	使用办公软件

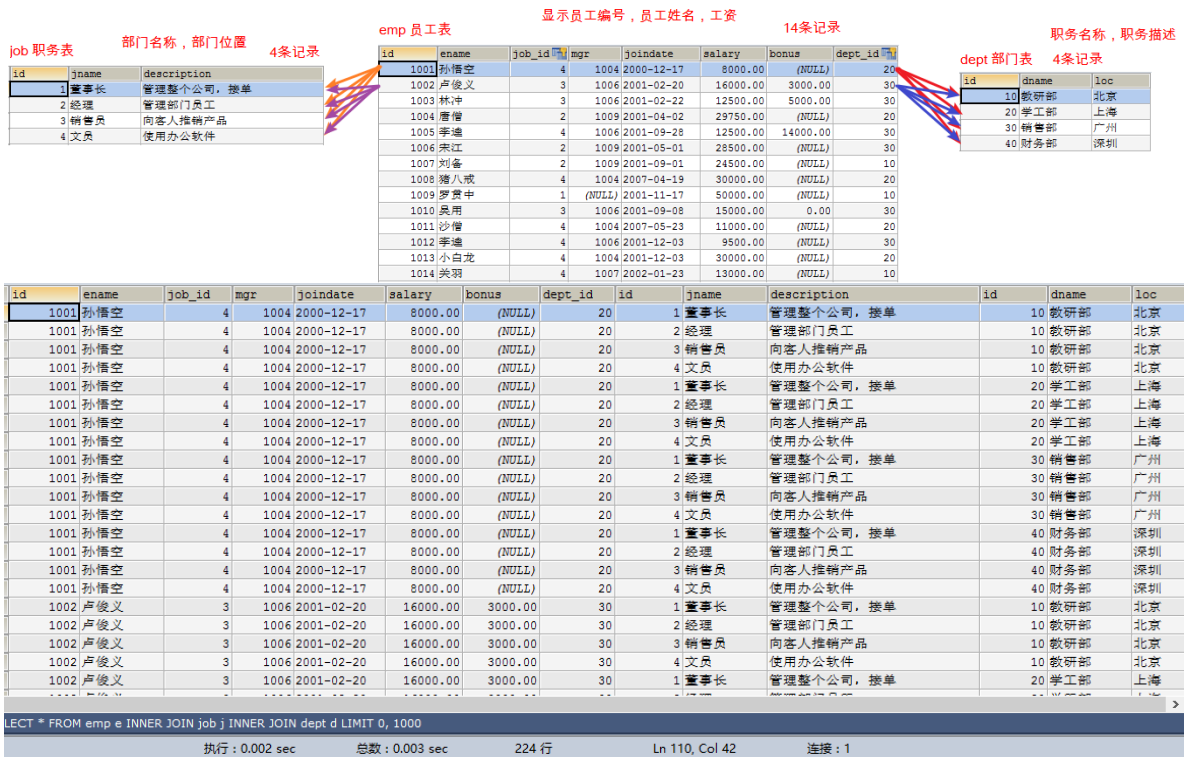
4.2.2 练习2

查询所有员工信息。显示员工编号，员工姓名，工资，职务名称，职务描述，部门名称，部门位置

具体操作：

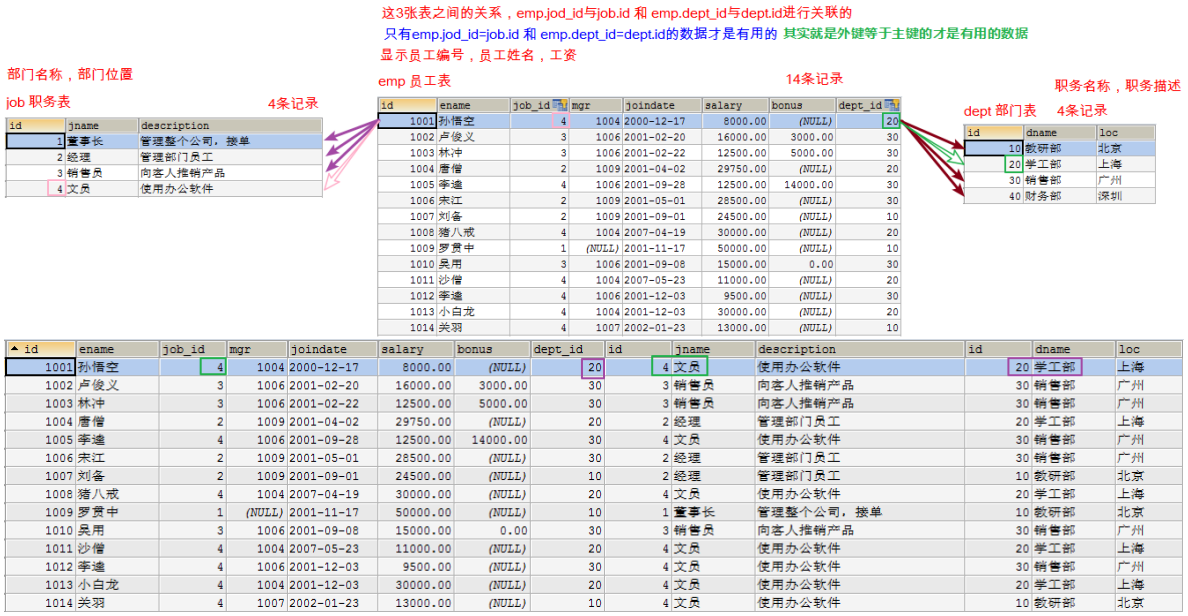
1. 确定要查询哪些表，emp e, job j, dept d

```
SELECT * FROM emp e INNER JOIN job j INNER JOIN dept d;
```



2. 确定表连接条件 e.job_id=j.id and e.dept_id=d.id

```
SELECT * FROM emp e INNER JOIN job j INNER JOIN dept d ON e.job_id=j.id AND e.dept_id=d.id;
```



3. 确定查询字段：员工编号，员工姓名，工资，职务名称，职务描述，部门名称，部门位置

```
SELECT e.`id`, e.`ename`, e.`salary`, j.`jname`, j.`description`, d.`dname`, d.`loc` FROM emp e INNER JOIN job j INNER JOIN dept d ON e.job_id=j.id AND e.dept_id=d.id;
```

id	ename	salary	jname	description	dname	loc
1009	罗贯中	50000.00	董事长	管理整个公司，接单	教研部	北京
1007	刘备	24500.00	经理	管理部门员工	教研部	北京
1014	关羽	13000.00	文员	使用办公软件	教研部	北京
1004	唐僧	29750.00	经理	管理部门员工	学工部	上海
1001	孙悟空	8000.00	文员	使用办公软件	学工部	上海
1008	猪八戒	30000.00	文员	使用办公软件	学工部	上海
1011	沙僧	11000.00	文员	使用办公软件	学工部	上海
1013	小白龙	30000.00	文员	使用办公软件	学工部	上海
1006	宋江	28500.00	经理	管理部门员工	销售部	广州
1002	卢俊义	16000.00	销售员	向客人推销产品	销售部	广州
1003	林冲	12500.00	销售员	向客人推销产品	销售部	广州
1010	吴用	15000.00	销售员	向客人推销产品	销售部	广州
1005	李逵	12500.00	文员	使用办公软件	销售部	广州
1012	李逵	9500.00	文员	使用办公软件	销售部	广州

4.2.3 练习3

查询所有员工信息。显示员工姓名，工资，职务名称，职务描述，部门名称，部门位置，工资等级

具体操作： 1. 确定要查询哪些表，emp e, job j, dept d, salarygrade s

```
    SELECT * FROM emp e INNER JOIN job j INNER JOIN dept d INNER JOIN salarygrade
s;
```

2. 确定表连接条件 e.job_id=j.id and e.dept_id=d.id and e.salary between s.losalary and hisalary

```
    SELECT * FROM emp e INNER JOIN job j INNER JOIN dept d INNER JOIN salarygrade
s ON e.job_id=j.id AND e.dept_id=d.id AND e.salary BETWEEN s.losalary AND
hisalary;
```

3. 确定查询字段：员工姓名，工资，职务名称，职务描述，部门名称，部门位置，工资等级

```
   SELECT e.`ename`, e.`salary`, j.`jname`, j.`description`, d.`dname`, d.`loc`, s.`grade` FROM emp e INNER JOIN job j INNER JOIN dept d INNER JOIN salarygrade s ON e.job_id=j.id AND e.dept_id=d.id AND e.salary BETWEEN s.losalary AND hisalary;
```

ename	salary	jname	description	dname	loc	grade
孙悟空	8000.00	文员	使用办公软件	学工部	上海	1
沙僧	11000.00	文员	使用办公软件	学工部	上海	1
李逵	9500.00	文员	使用办公软件	销售部	广州	1
林冲	12500.00	销售员	向客人推销产品	销售部	广州	2
李逵	12500.00	文员	使用办公软件	销售部	广州	2
关羽	13000.00	文员	使用办公软件	教研部	北京	2
卢俊义	16000.00	销售员	向客人推销产品	销售部	广州	3
吴用	15000.00	销售员	向客人推销产品	销售部	广州	3
唐僧	29750.00	经理	管理部门员工	学工部	上海	4
宋江	28500.00	经理	管理部门员工	销售部	广州	4
刘备	24500.00	经理	管理部门员工	教研部	北京	4
猪八戒	30000.00	文员	使用办公软件	学工部	上海	4
小白龙	30000.00	文员	使用办公软件	学工部	上海	4
罗贯中	50000.00	董事长	管理整个公司，接单	教研部	北京	5

4.2.3.1 多表查询规律总结

1. 不管我们查询几张表，表连接查询会产出笛卡尔积，我们需要消除笛卡尔积，拿到正确的数据。我们需要找到表与表之间通过哪个字段关联起来的（通常是`外键=主键`）
2. 消除笛卡尔积规律：2张表至少需要1个条件，3张表至少需要2个条件，4张表至少需要3个条件。（条件数量=表的数量-1），每张表都要参与进来
3. 多表连接查询步骤： 3.1. 确定要查询哪些表 3.2. 确定表连接条件 3.3. 确定查询字段

4.2.4 练习4

查询经理的信息。显示员工姓名，工资，职务名称，职务描述，部门名称，部门位置，工资等级

具体操作： 1. 确定要查询哪些表，emp e, job j, dept d, salarygrade s

```
SELECT * FROM emp e INNER JOIN job j INNER JOIN dept d INNER JOIN
salarygrade s;
```

2. 确定表连接条件 e.job_id=j.id and e.dept_id=d.id and e.salary between s.losalary and hisalary

```
SELECT * FROM emp e INNER JOIN job j INNER JOIN dept d INNER JOIN salarygrade
s ON e.job_id=j.id AND e.dept_id=d.id AND e.salary BETWEEN s.losalary AND
hisalary;
```

额外条件：只需要查询经理的信息（j.jname='经理'）

id	ename	job_id	mgr	joindate	salary	bonus	dept_id	id	jname	description	id	dname	loc	grade	losalary	hisalary
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20	2	经理	管理部门员工	20	学工部	上海	4	20010	30000
1006	宋江	2	1009	2001-05-01	28500.00	(NULL)	30	2	经理	管理部门员工	30	销售部	广州	4	20010	30000
1007	刘备	2	1009	2001-09-01	24500.00	(NULL)	10	2	经理	管理部门员工	10	教研部	北京	4	20010	30000

3. 确定查询字段：员工姓名，工资，职务名称，职务描述，部门名称，部门位置，工资等级

```
SELECT e.`ename`, e.`salary`, j.`jname`, j.`description`, d.`dname`, d.`loc`, s.`grade` FROM emp e INNER JOIN job j INNER JOIN dept d INNER JOIN salarygrade s ON e.job_id=j.id AND e.dept_id=d.id AND e.salary BETWEEN s.losalary AND hisalary AND j.jname='经理';
```

ename	salary	jname	description	dname	loc	grade
唐僧	29750.00	经理	管理部门员工	学工部	上海	4
宋江	28500.00	经理	管理部门员工	销售部	广州	4
刘备	24500.00	经理	管理部门员工	教研部	北京	4

4.2.5 练习5

查询出部门编号、部门名称、部门位置、部门人数

具体操作:

1. 去员工表中找到每个部门的人数和部门id

```
SELECT dept_id, COUNT(*) FROM emp GROUP BY dept_id;
```

查询出部门编号、部门名称、部门位置、部门人数

部门编号、部门名称、部门位置在部门表中
部门人数在员工表中统计

emp 员工表

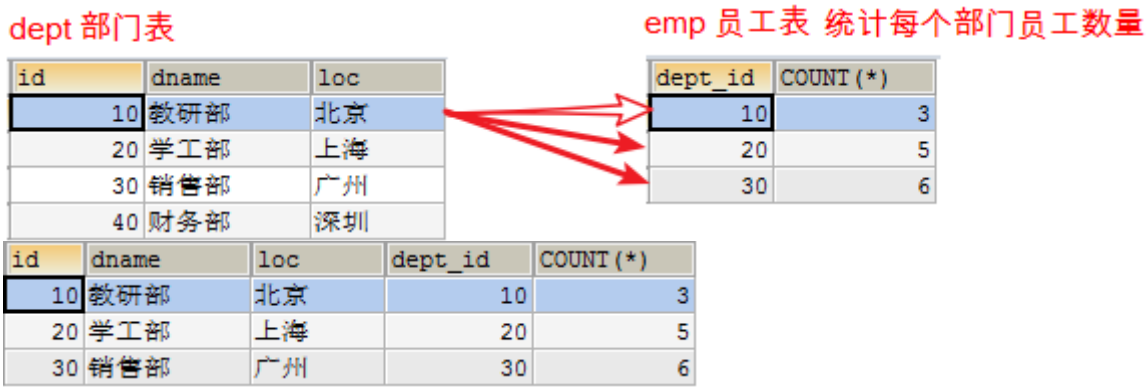
id	ename	job_id	mgr	joindate	salary	bonus	dept_id
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20
1005	李逵	4	1006	2001-09-28	12500.00	14000.00	30
1006	宋江	2	1009	2001-05-01	28500.00	(NULL)	30
1007	刘备	2	1009	2001-09-01	24500.00	(NULL)	10
1008	猪八戒	4	1004	2007-04-19	30000.00	(NULL)	20
1009	罗贯中	1	(NULL)	2001-11-17	50000.00	(NULL)	10
1010	吴用	3	1006	2001-09-08	15000.00	0.00	30
1011	沙僧	4	1004	2007-05-23	11000.00	(NULL)	20
1012	李逵	4	1006	2001-12-03	9500.00	(NULL)	30
1013	小白龙	4	1004	2001-12-03	30000.00	(NULL)	20
1014	关羽	4	1007	2002-01-23	13000.00	(NULL)	10

SELECT dept_id, COUNT(*) FROM emp GROUP BY dept_id;
emp 员工表 统计每个部门员工数量

dept_id	COUNT(*)
10	3
20	5
30	6

2. 再和部门表连接查询

```
SELECT * FROM dept d INNER JOIN (SELECT dept_id, COUNT(*) FROM emp GROUP BY dept_id) e ON e.dept_id=d.`id`;
```



id	dname	loc	dept_id	COUNT(*)
10	教研部	北京	10	3
20	学工部	上海	20	5
30	销售部	广州	30	6

3. 显示对应的字段

```
   SELECT d.`id`, d.dname, d.`loc`, e.total 部门人数 FROM dept d INNER JOIN
(SELECT dept_id, COUNT(*) total FROM emp GROUP BY dept_id) e ON
e.dept_id=d.`id`;
```

因为要取出COUNT(*)这个字段，所以给这个字段取别名，后面使用表名.字段名取出

id	dname	loc	dept_id	COUNT(*)
10	教研部	北京	10	3
20	学工部	上海	20	5
30	销售部	广州	30	6

最终效果：

```
SELECT * FROM dept d INNER JOIN (
SELECT dept_id, COUNT(*) total FROM emp GROUP BY dept_id) e
ON e.dept_id=d.`id`;
```

id	dname	loc	dept_id	total
10	教研部	北京	10	3
20	学工部	上海	20	5
30	销售部	广州	30	6

id	dname	loc	部门人数
10	教研部	北京	3
20	学工部	上海	5
30	销售部	广州	6

第5章 事务安全

目标

完成转账案例

5.1 事务的应用场景说明

在实际的业务开发中，有些业务操作要多次访问数据库。一个业务要发送多条SQL语句给数据库执行。需要将多次访问数据库的操作视为一个整体来执行，要么都执行成功，要么都执行失败。如果其中有一条SQL语句失败，就进行事务的回滚，所有的SQL语句全部执行失败。 例如: 张三给李四转账，张三账号减钱，李四账号加钱

```sql
-- 创建数据表
CREATE TABLE account (
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(10),
    balance DOUBLE
);

-- 添加数据
INSERT INTO account (NAME, balance) VALUES ('张三', 1000), ('李四', 1000);
```

模拟张三给李四转500元钱，一个转账的业务操作最少要执行下面的2条语句：

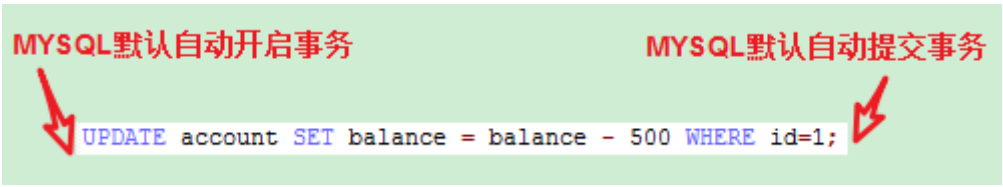
1. 张三账号-500
2. 李四账号+500

```sql
-- 1. 张三账号-500
UPDATE account SET balance = balance - 500 WHERE id=1;
-- 2. 李四账号+500
UPDATE account SET balance = balance + 500 WHERE id=2;
```

假设当张三账号上-500元,服务器崩溃了。李四的账号并没有+500元，数据就出现问题了。我们需要保证其中一条SQL语句出现问题，整个转账就算失败。只有两条SQL都成功了转账才算成功。这个时候就需要用到事务

5.2 操作事务

MySQL默认开启自动提交事务,意思是MySQL的每一条DML(增删改)语句都是一个单独的事务，每条语句都会自动开启一个事务，执行完毕自动提交事务。



1. 将金额重置为1000

id	NAME	balance
1	张三	1000
2	李四	1000

2. 执行以下SQL语句

```sql
UPDATE account SET balance = balance - 500 WHERE id=1;
```

3. 使用SQLYog查看数据库：发现数据已经改变

id	name	balance
1	张三	500
2	李四	1000

MYSQL中可以通过以下两种方式进行控制事务的操作：

`1.手动提交事务`

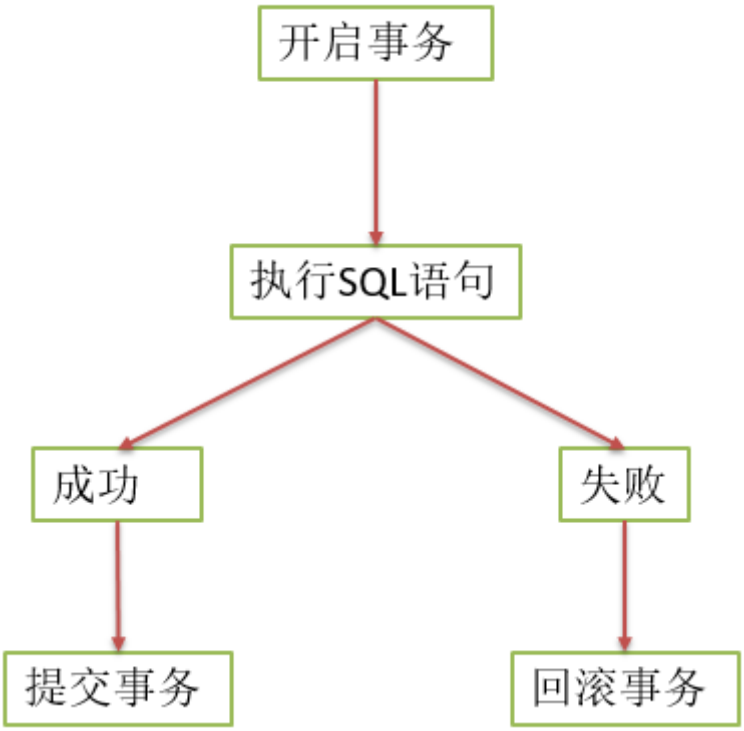
`2.自动提交事务`

5.2.1 手动提交事务

事务有关的SQL语句：

SQL语句	描述
start transaction;	开启事务
commit;	提交事务
rollback;	回滚事务

手动提交事务使用步骤： 第1种情况：开启事务 -> 执行SQL语句 -> 成功 -> 提交事务 第2种情况：开启事务 -> 执行SQL语句 -> 失败 -> 回滚事务



案例演示1：模拟张三给李四转500元钱（成功） 目前数据库数据如下：

id	name	balance
1	张三	1000
2	李四	1000

1. 使用DOS控制台进入MySQL
2. 执行以下SQL语句： `1.开启事务`， `2.张三账号-500`， `3.李四账号+500`

```
START TRANSACTION;
UPDATE account SET balance = balance - 500 WHERE id=1;
UPDATE account SET balance = balance + 500 WHERE id=2;
```

```
mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE account SET balance = balance - 500 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance + 500 WHERE id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

3. 使用SQLYog查看数据库：发现数据并没有改变

id	name	balance
1	张三	1000
2	李四	1000

4. 在控制台执行`commit`提交任务：

```
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

5. 使用SQLYog查看数据库：发现数据改变

id	NAME	balance
1	张三	500
2	李四	1500

案例演示2：模拟张三给李四转500元钱（失败） 目前数据库数据如下：

id	NAME	balance
1	张三	500
2	李四	1500

1. 在控制台执行以下SQL语句：`1.开启事务`， `2.张三账号-500`

```
START TRANSACTION;
UPDATE account SET balance = balance - 500 WHERE id=1;
```

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE account SET balance = balance - 500 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

2. 使用SQLYog查看数据库：发现数据并没有改变

id	NAME	balance
1	张三	500
2	李四	1500

3. 在控制台执行`rollback`回滚事务：

```
mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

4. 使用SQLYog查看数据库：发现数据没有改变

id	NAME	balance
1	张三	500
2	李四	1500

> 总结: 如果事务中SQL语句没有问题，commit提交事务，会对数据库数据的数据进行改变。 如果事务中SQL语句有问题，rollback回滚事务，会回退到开启事务时的状态。

5.2.2 自动提交事务

通过修改MySQL全局变量"autocommit"，取消自动提交事务

1. 使用SQL语句：`show variables like '%commit%';`查看MySQL是否开启自动提交事务

```
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| autocommit                    | ON    |
| innodb_commit_concurrency     | 0     |
| innodb_flush_log_at_trx_commit | 1     |
+-------------------------------+-------+
```

0:OFF（关闭自动提交） 1:ON（开启自动提交）

2. 取消自动提交事务，设置自动提交的参数为OFF，执行SQL语句：`set autocommit = 0;`

```
mysql> set autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like '%commit%';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| autocommit                    | OFF   |
| innodb_commit_concurrency     | 0     |
| innodb_flush_log_at_trx_commit | 1     |
+-------------------------------+-------+
3 rows in set (0.00 sec)
```

3. 在控制台执行以下SQL语句：张三-500

```
UPDATE account SET balance = balance - 500 WHERE id=1;
```

```
mysql> UPDATE account SET balance = balance - 500 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

4. 使用SQLYog查看数据库，发现数据并没有改变

id	name	balance
1	张三	1000
2	李四	1000

5. 在控制台执行`commit`提交任务

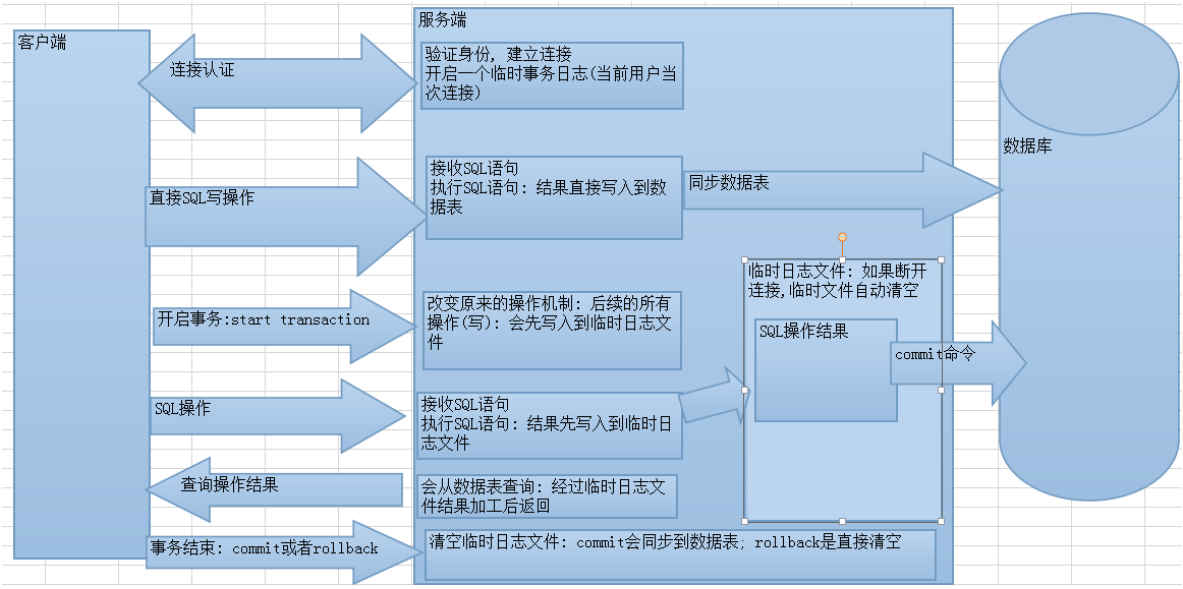
```
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

6. 使用SQLYog查看数据库，发现数据改变

id	name	balance
1	张三	500
2	李四	1000

5.3 事务原理

事务开启之后，所有的操作都会临时保存到事务日志，事务日志只有在得到`commit`命令才会同步到数据表中，其他任何情况都会清空事务日志(rollback，断开连接)



5.4 回滚点

在某些操作成功之后，后续的操作有可能成功有可能失败，但是不管成功还是失败，前面操作都已经成功，可以在当前成功的位置设置一个回滚点。可以供后续失败操作返回到该位置，而不是返回所有操作，这个点称之为回滚点。

设置回滚点语法：`savepoint 回滚点名字;` 回到回滚点语法：`rollback to 回滚点名字;`

具体操作：

1. 将数据重置为1000

```
mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |    1000 |
|  2 | 李四 |    1000 |
+----+------+---------+
```

2. 开启事务

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

3. 让张三账号减3次钱

```
UPDATE account SET balance = balance - 10 WHERE id=1;
UPDATE account SET balance = balance - 10 WHERE id=1;
UPDATE account SET balance = balance - 10 WHERE id=1;
```

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE account SET balance = balance - 10 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

4. 设置回滚点：`savepoint abc;`

```
mysql> savepoint abc;
Query OK, 0 rows affected (0.00 sec)
```

5. 让张三账号减4次钱

```
UPDATE account SET balance = balance - 10 WHERE id=1;
UPDATE account SET balance = balance - 10 WHERE id=1;
UPDATE account SET balance = balance - 10 WHERE id=1;
UPDATE account SET balance = balance - 10 WHERE id=1;
```

```
mysql> savepoint abc;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE account SET balance = balance - 10 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

6. 回到回滚点：`rollback to abc;`

```
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |     970 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)
```

7. 分析过程

```
+----+------+---------+
|  1 | 张三 |    1000 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE account SET balance = balance - 10 WHERE id=1; -> 990
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1; -> 980
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1; -> 970
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> savepoint abc;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE account SET balance = balance - 10 WHERE id=1; -> 960
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1; -> 950
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1; -> 940
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE account SET balance = balance - 10 WHERE id=1; -> 930
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

回到回滚点，970

mysql> rollback to abc;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |     970 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)
```

> 总结：设置回滚点可以让我们在失败的时候回到回滚点，而不是回到事务开启的时候。

5.5 事务的四大特性

5.5.1 事务的四大特性

事务特性	含义
原子性（Atomicity）	事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
一致性（Consistency）	事务前后数据的完整性必须保持一致
隔离性（Isolation）	是指多个用户并发访问数据库时，一个用户的事务不能被其它用户的事务所干扰，多个并发事务之间数据要相互隔离，不能相互影响。
持久性（Durability）	指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响

5.5.2 事务的隔离级别

事务在操作时的理想状态：多个事务之间互不影响，如果隔离级别设置不当就可能引发并发读取问题。

并发读取问题	含义
脏读	一个事务读取到了另一个事务中尚未提交的数据
不可重复读	一个事务中两次读取的数据内容不一致，要求的是一个事务中多次读取时数据是一致的，这是事务update时引发的问题
幻读	一个事务中两次读取的数据的数量不一致，要求在一个事务多次读取的数据的数量是一致的，这是insert或delete时引发的问题

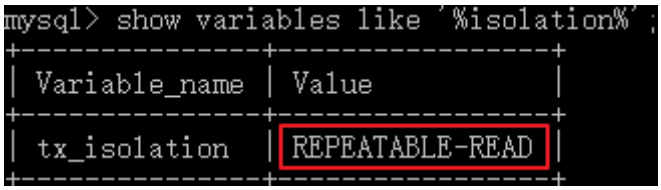
MySQL数据库有四种隔离级别：上面的级别最低，下面的级别最高。“是”表示会出现这种问题，“否”表示不会出现这种问题。

级别	名字	隔离级别	脏读	不可重复读	幻读	数据库默认隔离级别
1	读未提交	read uncommitted	是	是	是	
2	读已提交	read committed	否	是	是	Oracle和SQL Server
3	可重复读	repeatable read	否	否	是	MySQL
4	串行化	serializable	否	否	否	

MySQL事务隔离级别相关的命令

1. 查询全局事务隔离级别

```
show variables like '%isolation%';
-- 或
select @@tx_isolation;
```



2. 设置事务隔离级别，需要退出MSQL再进入MYSQL才能看到隔离级别的变化

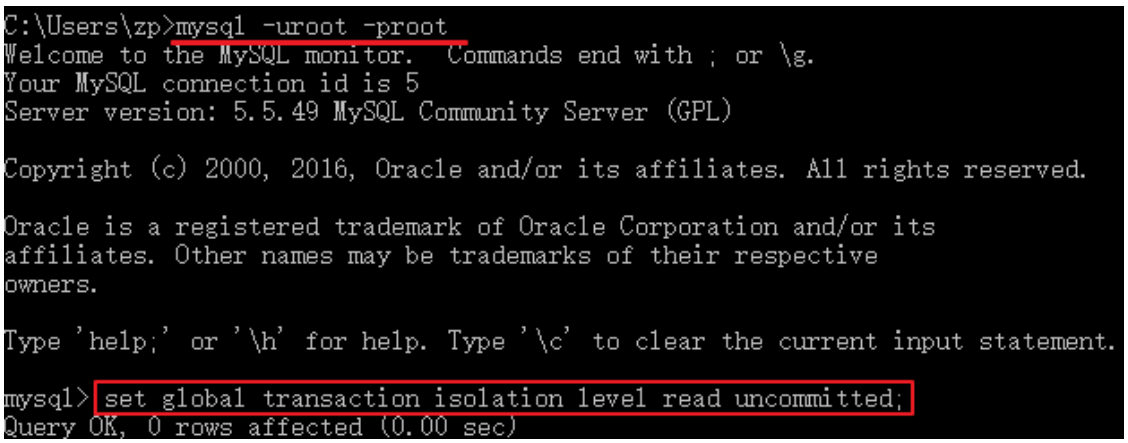
```
set global transaction isolation level 级别字符串;
-- 如:
set global transaction isolation level read uncommitted;
```

5.5.2.1 脏读的演示

将数据进行恢复：`UPDATE account SET balance = 1000;`

1. 打开A窗口登录MySQL，设置全局的隔离级别为最低

```
mysql -uroot -proot
set global transaction isolation level read uncommitted;
```

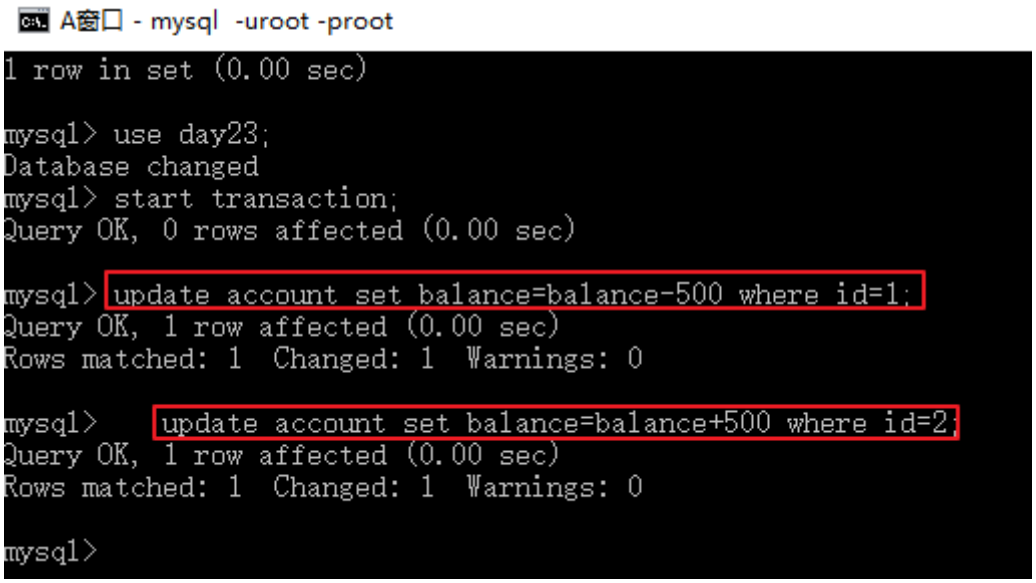


2. 打开B窗口,AB窗口都开启事务

```
use day23;
start transaction;
```

3. A窗口更新2个人的账户数据，未提交

```
update account set balance=balance-500 where id=1;
update account set balance=balance+500 where id=2;
```



4. B窗口查询账户

```
select * from account;
```

5. A窗口回滚

```
rollback;
```

A窗口 - mysql -uroot -proot

```
1 row in set (0.00 sec)

mysql> use day23;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance=balance-500 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>    update account set balance=balance+500 where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

6. B窗口查询账户，钱没了

```
mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |    1000 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)
```

A窗口的事务回滚了，这边的钱也跟着变化了，钱没了

脏读非常危险的，比如张三向李四购买商品，张三开启事务，向李四账号转入500块，然后打电话给李四说钱已经转了。李四一查询钱到账了，发货给张三。张三收到货后回滚事务，李四的再查看钱没了。

解决脏读的问题：将全局的隔离级别进行提升 将数据进行恢复：`UPDATE account SET balance = 1000;`

1. 在A窗口设置全局的隔离级别为`read committed`

```
set global transaction isolation level read committed;
```

选择A窗口 - mysql -uroot -proot

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql> set global transaction isolation level read committed;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

2. B窗口退出MySQL，B窗口再进入MySQL

B窗口 - mysql -uroot -proot

```
mysql> exit
Bye

C:\Users\zp>mysql -uroot -proot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.5.49 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

3. AB窗口同时开启事务

A窗口 - mysql -uroot -proot

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

开启事务

B窗口 - mysql -uroot -proot

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

开启事务

4. A更新2个人的账户，未提交

```
update account set balance=balance-500 where id=1;
update account set balance=balance+500 where id=2;
```

A窗口 - mysql -uroot -proot

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance=balance-500 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set balance=balance+500 where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

5. B窗口查询账户

B窗口 - mysql -uroot -proot

```
mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |    1000 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)

mysql>
```

没有读取到另一个事务未提交的事务

6. A窗口commit提交事务

A窗口 - mysql -uroot -proot

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance=balance-500 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set balance=balance+500 where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

7. B窗口查看账户

B窗口 - mysql -uroot -proot

```
mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |    1000 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)

mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |     500 |
|  2 | 李四 |    1500 |
+----+------+---------+
2 rows in set (0.00 sec)

mysql>
```

另一个事务提交后的数据才读取到

> 结论：read committed的方式可以避免脏读的发生

5.5.2.2 不可重复读的演示

将数据进行恢复：`UPDATE account SET balance = 1000;`

1. 开启A窗口

```
set global transaction isolation level read committed;
```

A窗口 - mysql -uroot -proot

```
mysql> set global transaction isolation level read committed;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

2. 开启B窗口，在B窗口开启事务

```
start transaction;
select * from account;
```

```
mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |    1000 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)
```

3. 在A窗口开启事务，并更新数据

```
start transaction;
update account set balance=balance+500 where id=1;
commit;
```

A窗口 - mysql -uroot -proot

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance=balance+500 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

4. B窗口查询

```
select * from account;
```

B窗口 - mysql -uroot -proot

```
mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |    1000 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)

mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |    1500 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)

mysql>
```

两次查询输出的结果不同，到底哪次是对的？

两次查询输出的结果不同，到底哪次是对的？不知道以哪次为准。 很多人认为这种情况就对了，无须困惑，当然是后面的为准。我们可以考虑这样一种情况，比如银行程序需要将查询结果分别输出到电脑屏幕和发短信给客户，结果在一个事务中针对不同的输出目的地进行的两次查询不一致，导致文件和屏幕中的结果不一致，银行工作人员就不知道以哪个为准了。

解决不可重复读的问题：将全局的隔离级别进行提升为：`repeatable read` 将数据进行恢复：`UPDATE account SET balance = 1000;`

1. A窗口设置隔离级别为：`repeatable read`

```
set global transaction isolation level repeatable read;
```

```
mysql> set global transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00 sec)
```

2. B窗口退出MySQL，B窗口再进入MySQL

```
start transaction;
select * from account;
```

B窗口 - mysql -uroot -proot

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三 |    1000 |
|  2 | 李四 |    1000 |
+----+------+---------+
2 rows in set (0.00 sec)
```

3. A窗口更新数据

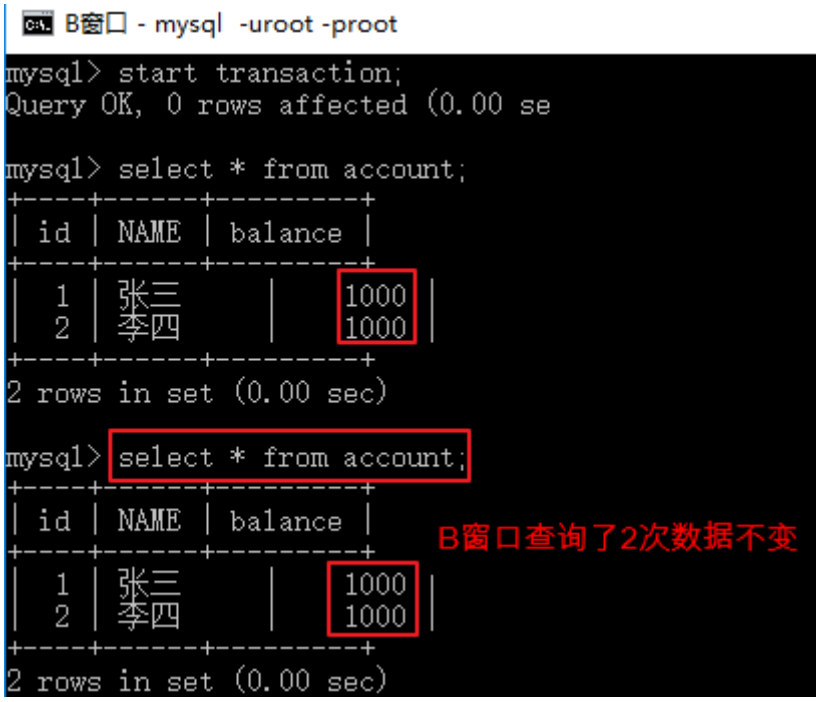
```
start transaction;
update account set balance=balance+500 where id=1;
commit;
```

A窗口 - mysql -uroot -proot

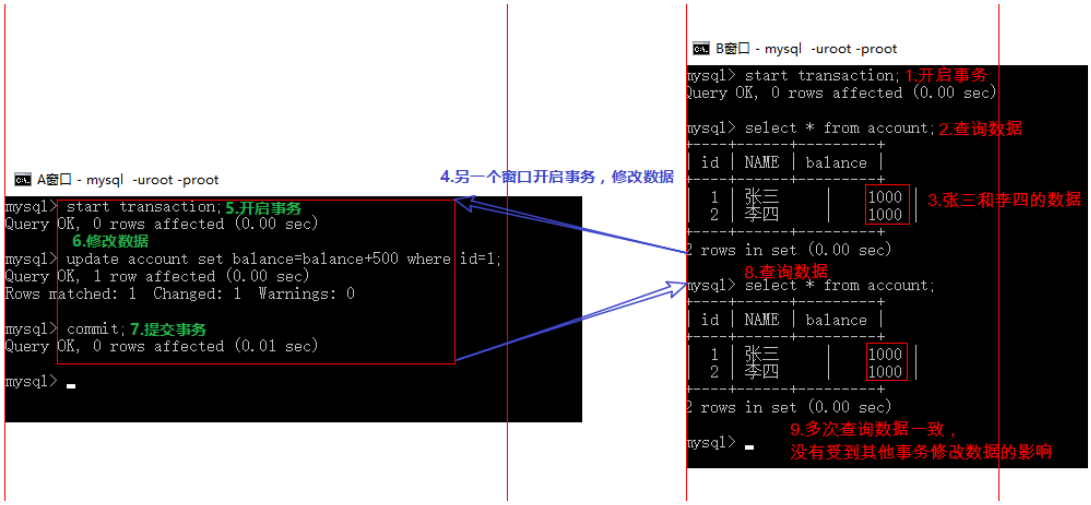
```
mysql> set global transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00 sec)

mysql> use day23;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance=balance+500 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

4. B窗口查询

```
select * from account;
```



> 结论：同一个事务中为了保证多次查询数据一致，必须使用`repeatable read`隔离级别

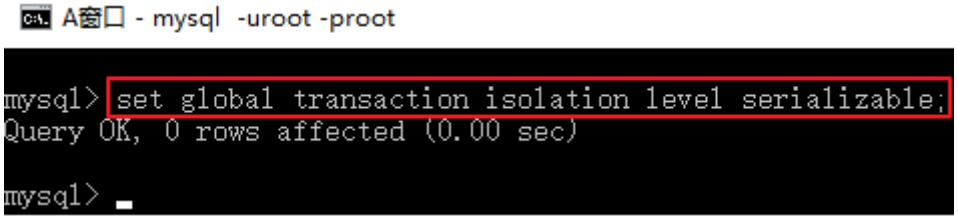


5.5.2.3 幻读的演示

在MySQL中无法看到幻读的效果。但我们可以将事务隔离级别设置到最高，以挡住幻读的发生 将数据进行恢复：`UPDATE account SET balance = 1000;`

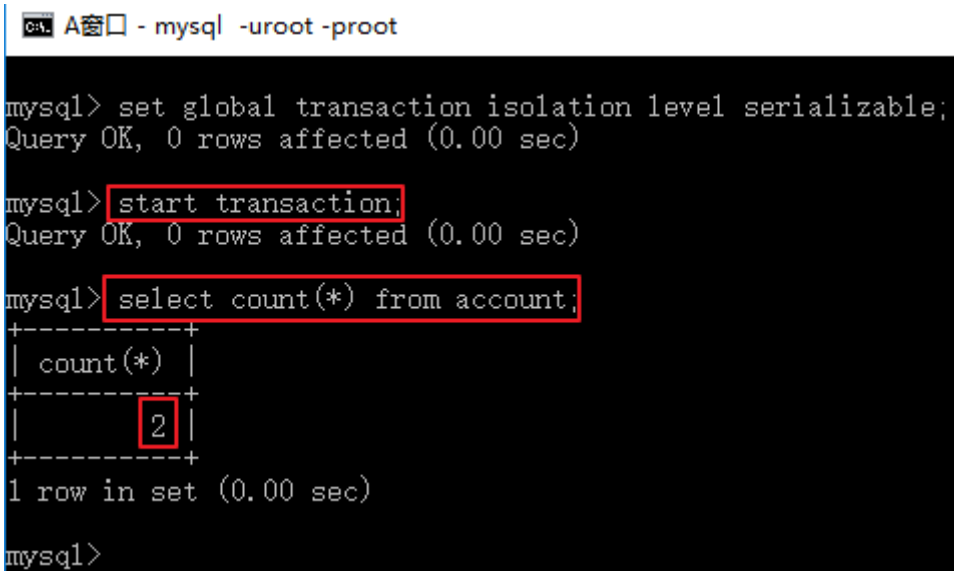
1. 开启A窗口

```
set global transaction isolation level serializable; -- 设置隔离级别为最高
```



2. A窗口退出MySQL，A窗口重新登录MySQL

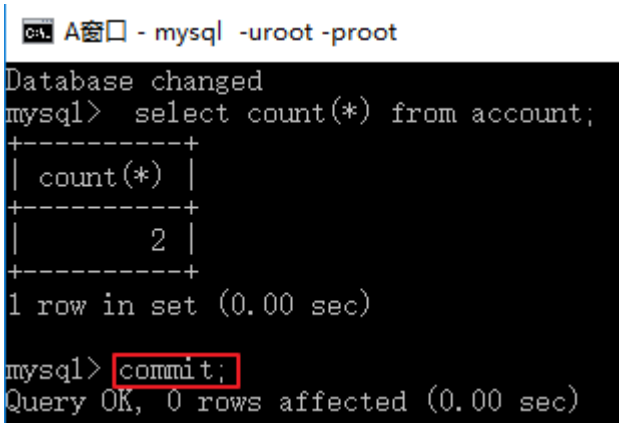
```
start transaction;
select count(*) from account;
```



3. 再开启B窗口，登录MySQL
4. 在B窗口中开启事务，添加一条记录

```
start transaction; -- 开启事务
insert into account (name,balance) values ('LaoWang', 500);
```

5. 在A窗口中commit提交事务，B窗口中insert语句会在A窗口事务提交后立马运行



6. 在A窗口中接着查询，发现数据不变

```
select count(*) from account;
```

A窗口 - mysql -uroot -proot

```
mysql>  select count(*) from account;
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>  select count(*) from account;
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

7. B窗口中commit提交当前事务

B窗口 - mysql -uroot -proot

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into account (name,balance) values ('LaoWang', 500);
Query OK, 1 row affected (12.86 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

8. A窗口就能看到最新的数据

A窗口 - mysql -uroot -proot

```
mysql>  select count(*) from account;
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>  select count(*) from account;
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)

mysql>  select count(*) from account;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

> 结论:使用serializable隔离级别，一个事务没有执行完，其他事务的SQL执行不了，可以挡住幻读

小结

> 所谓事务其实就是将多个操作看成同一个逻辑操作，要么全部成功，要么全部失败
>
> 企业中使用`read commited`的场景比较多，因为使用`repeatable read`在没有索引的情况下性能较差

第6章 DCL

目标

能够看懂DCL命令，并按照自己的需求修改命令

我们现在默认使用的都是root用户，超级管理员，拥有全部的权限。但是，一个公司里面的数据库服务器上面可能同时运行着很多个项目的数据库。所以，我们应该可以根据不同的项目建立不同的用户，分配不同的权限来管理和维护数据库。

6.1 创建用户

`CREATE USER '用户名'@'主机名' IDENTIFIED BY '密码';` **关键字说明：**

```
  1. `用户名`：将创建的用户名
  2. `主机名`：指定该用户在哪个主机上可以登陆，如果是本地用户可用localhost，如果想让该用户可以从任意远程主机登陆，可以使用通配符%
  3. `密码`：该用户的登陆密码，密码可以为空，如果为空则该用户可以不需要密码登陆服务器
```

具体操作：

```
-- user1用户只能在localhost这个IP登录mysql服务器
CREATE USER 'user1'@'localhost' IDENTIFIED BY '123';
-- user2用户可以在任何电脑上登录mysql服务器
CREATE USER 'user2'@'%' IDENTIFIED BY '123';
```

6.2 授权用户

用户创建之后，基本没什么权限！需要给用户授权

```
D:\mysql\mysql-5.7.27-winx64\bin>mysql -u user1 -p   使用新创建的用户登录到数据库
Enter password: ***
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;   使用show databases 查看数据库，并没有我们要找的业务数据库
+--------------------+
| Database           |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.00 sec)

mysql>
```

授权格式：`GRANT 权限1, 权限2... ON 数据库名.表名 TO '用户名'@'主机名';` **关键字说明**：

1. `GRANT` 授权关键字
2. 授予用户的权限，如`SELECT`，`INSERT`，`UPDATE`等。如果要授予所的权限则使用`ALL`
3. `数据库名.表名`：该用户可以操作哪个数据库的哪些表。如果要授予该用户对所有数据库和表的相应操作权限则可用*表示，如`*.*`
4. `'用户名'@'主机名'`：给哪个用户授权

具体操作：

1. 给user1用户分配对test这个数据库操作的权限

```
GRANT CREATE,ALTER,DROP,INSERT,UPDATE,DELETE,SELECT ON test.* TO
'user1'@'localhost';
```

```
mysql> GRANT CREATE,ALTER,DROP,INSERT,UPDATE,DELETE,SELECT ON test.* TO 'user1'@'localhost';
Query OK, 0 rows affected (0.01 sec)   使用root用户给 user1用户赋予test数据库的操作权限

mysql> exit
Bye

D:\mysql\mysql-5.7.27-winx64\bin>mysql -u user1 -p   登录user1用户
Enter password: ***
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;   使用 show databases 可以查看到test数据库
+--------------------+
| Database           |
+--------------------+
| information_schema |
| test               |
+--------------------+
```

```
GRANT ALL ON *.* TO 'user2'@'%';
```

```
mysql>
mysql> GRANT ALL ON *.* TO 'user2'@'%';
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye

D:\mysql\mysql-5.7.27-winx64\bin>mysql -u user2 -p
Enter password: ***
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| db1                |
| mysql              |
| performance_schema |
| sys                |
| test               |
| test1              |
+--------------------+
```

使用root用户给user2用户赋予全部数据库的全部权限

登录user2用户

使用 show databases 命令可以查看到所有的数据库

6.3 撤销授权(了解)

`REVOKE 权限1, 权限2... ON 数据库.表名 FROM '用户名'@'主机名';`

具体操作:

- 撤销user1用户对test操作的权限

```
REVOKE ALL ON test.* FROM 'user1'@'localhost';
```

```
mysql>
mysql> REVOKE ALL ON test.* FROM 'user1'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> exit
Bye

D:\mysql\mysql-5.7.27-winx64\bin>mysql -uuser1 -p
Enter password: ***
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 5.7.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.00 sec)

mysql>
```

使用root用户给user1用户的所有权限撤销

登录user1用户

业务数据库看不到了

6.4 查看权限(了解)

`SHOW GRANTS FOR '用户名'@'主机名';` **具体操作:**

- 查看user1用户的权限

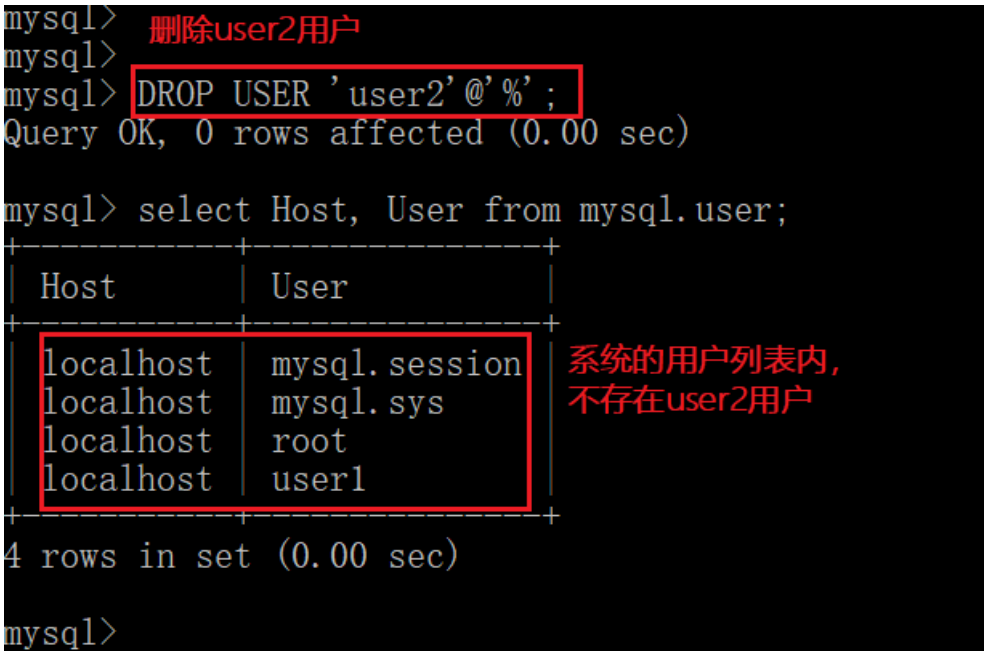
```
SHOW GRANTS FOR 'user1'@'localhost';
```

5.5 删除用户(了解)

`DROP USER '用户名'@'主机名';` **具体操作:**

- 删除user2

```
DROP USER 'user2'@'%';
```



6.6 修改用户密码(了解)

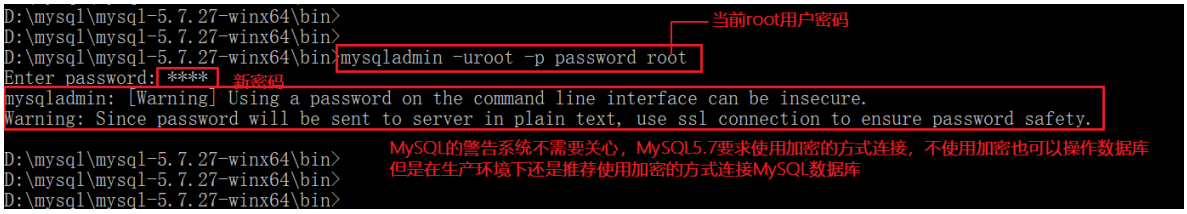
6.6.1 修改管理员密码

`mysqladmin -uroot -p password 新密码 -- 新密码不需要加上引号`

> 注意：需要在未登陆MySQL的情况下操作。

具体操作：

```
mysqladmin -uroot -p password 123456
输入老密码
```



6.6.2 修改普通用户密码

`set password for '用户名'@'主机名' = password('新密码');`

> 注意：需要在登陆MySQL的情况下操作。

具体操作：

```
set password for 'user1'@'localhost' = password('666666');
```