MySQL多表查询与事务安全

学习目标

1. 能够理解多表查询的规律
2. 能够使用内连接进行多表查询
3. 能够使用左外连接和右外连接进行多表查询
4. 能够使用子查询进行多表查询
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 事务安全

1.1 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，要么所有的SQL语句全部执行成功。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

```
-- 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

```
-- 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

1.2 操作事务

MYSQL中可以有两种方式进行事务的操作：`1.手动提交事务`，`2.自动提交事务`

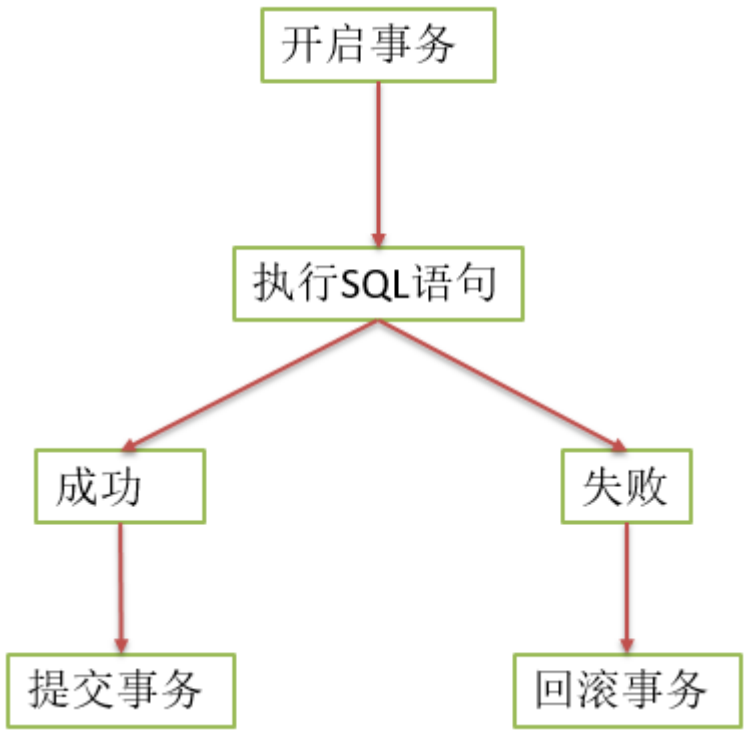
1.2.1 手动提交事务

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>
```

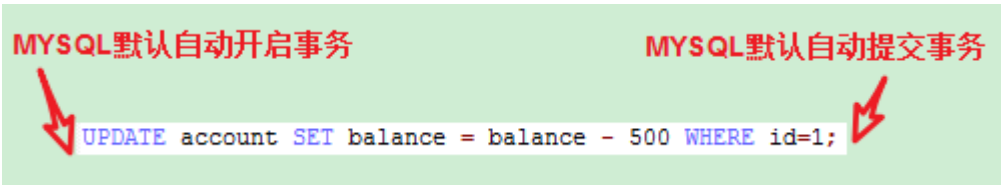
4. 使用SQLYog查看数据库：发现数据没有改变

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。

1.2.2 自动提交事务

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，MySQL默认开始自动提交事务



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

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

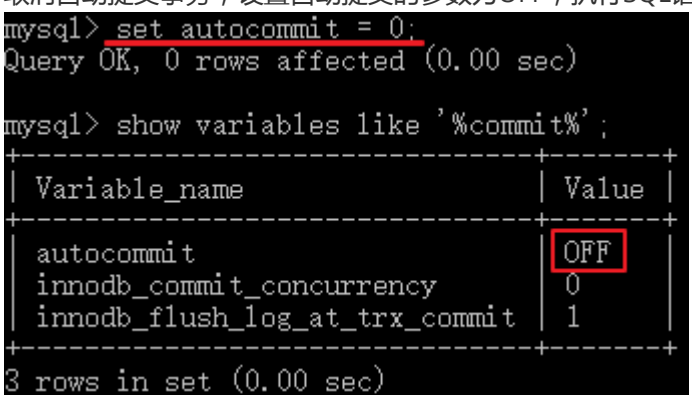
3. 使用SQLYog查看数据库：发现数据已经改变

id	NAME	balance
1	张三	500
2	李四	1500

通过修改MySQL全局变量"autocommit"，取消自动提交事务 使用SQL语句：`show variables like '%commit%';` 查看MySQL是否开启自动提交事务

0:OFF（关闭自动提交） 1:ON（开启自动提交）

4. 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;`



5. 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
```

6. 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

7. 在控制台执行`commit`提交任务

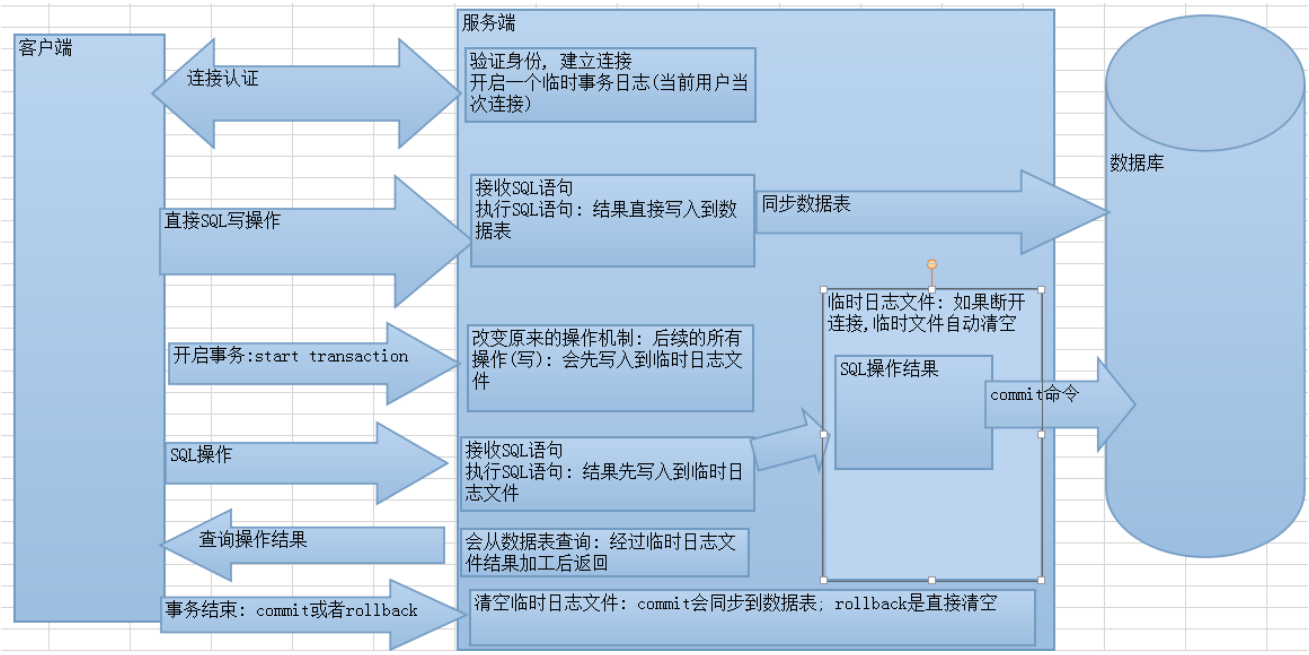
```
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

8. 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

1.3 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)



1.4 回滚点

在某些成功的操作完成之后，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. 将数据还原到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

mysql> rollback to abc;   回到回滚点，970
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。

1.5 事务的四大特性

1.5.1 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

1.5.2 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访问问题。

并发访问的问题	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

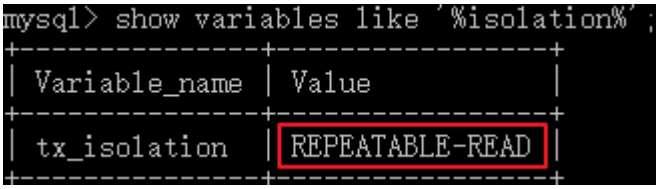
MySQL数据库有四种隔离级别：上面的级别最低，下面的级别最高。“是”表示会出现这种问题，“否”表示不会出现这种问题。

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

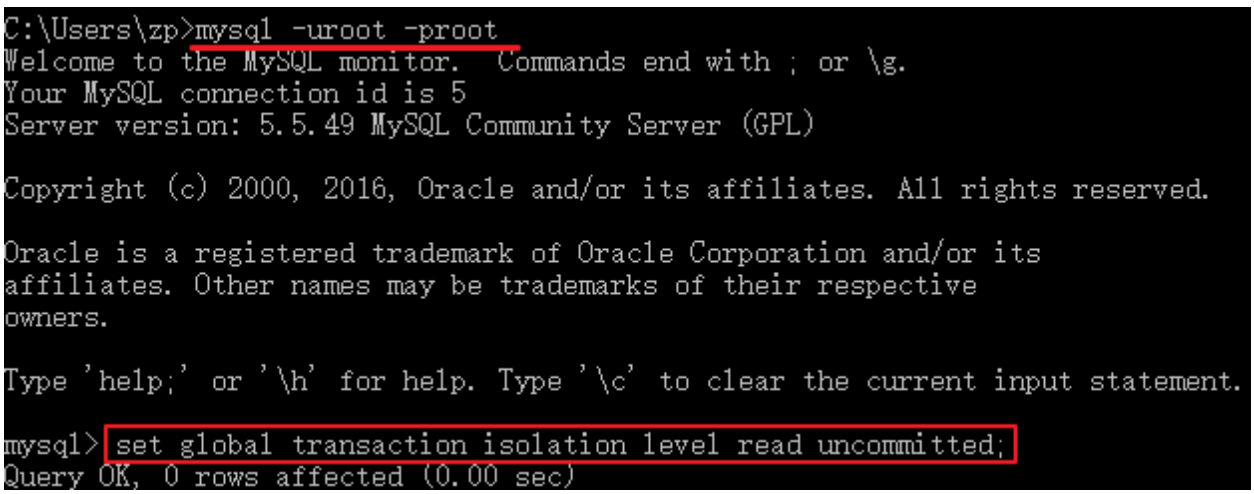
```
set global transaction isolation level 级别字符串;
-- 如:
set global transaction isolation level read uncommitted;
```

1.5.2.1 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;
```

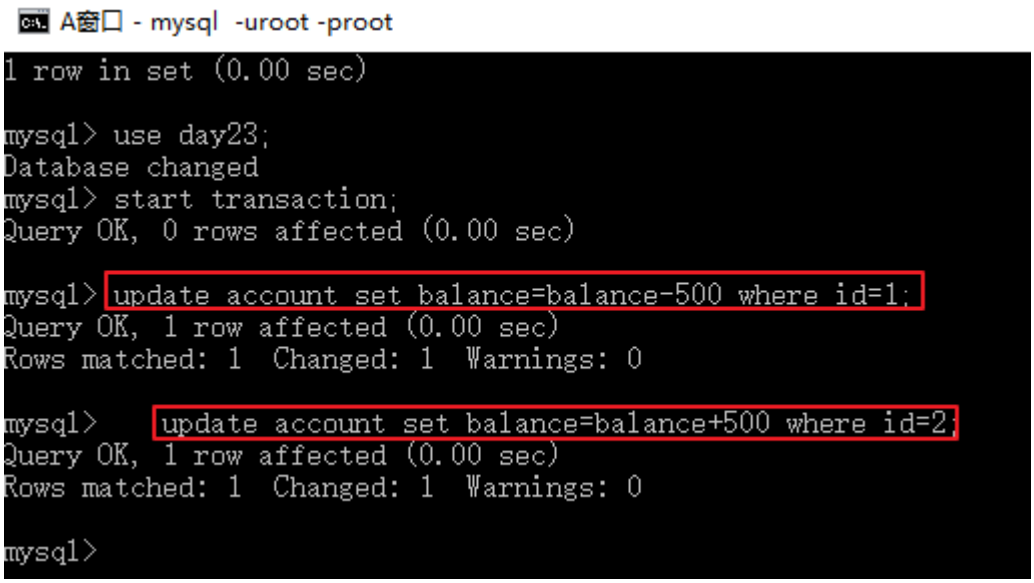


2. 打开B窗口,AB窗口都开启事务

```
use day23;

start transaction;
```

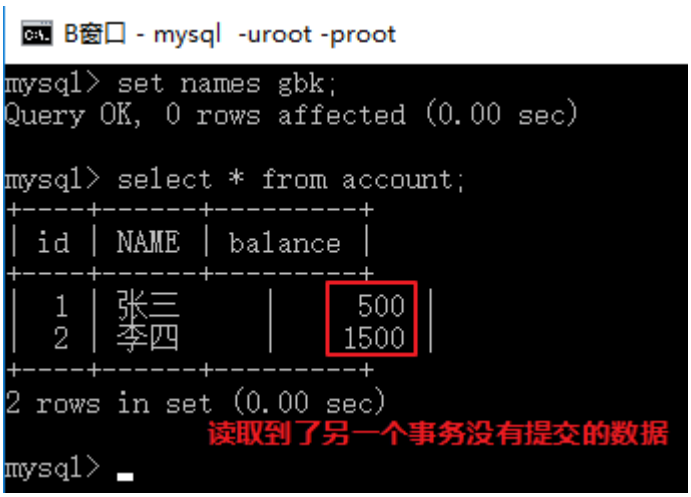
3. A窗口更新2个人的账户数据，未提交

```
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

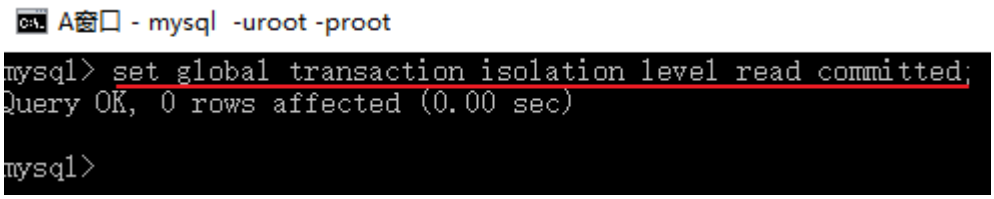
> 结论：read committed的方式可以避免脏读的发生

1.5.2.2 不可重复读的演示

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

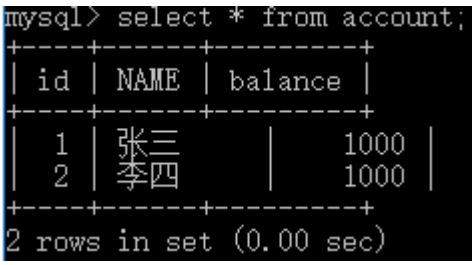
1. 开启A窗口

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



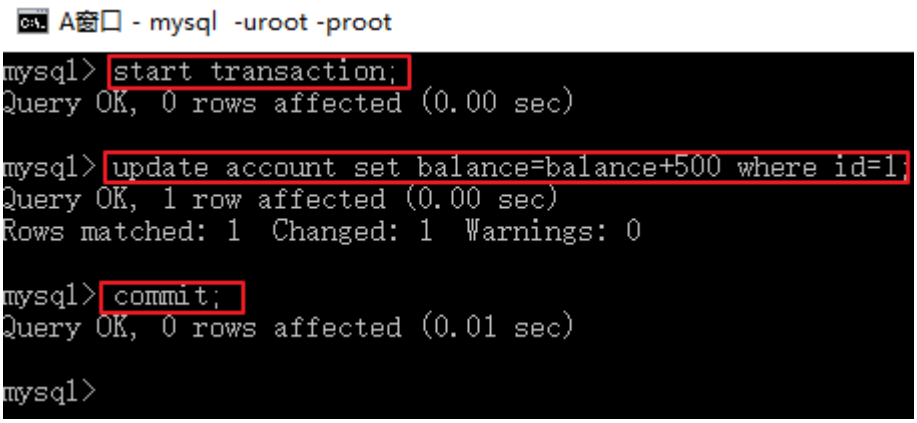
2. 开启B窗口，在B窗口开启事务

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



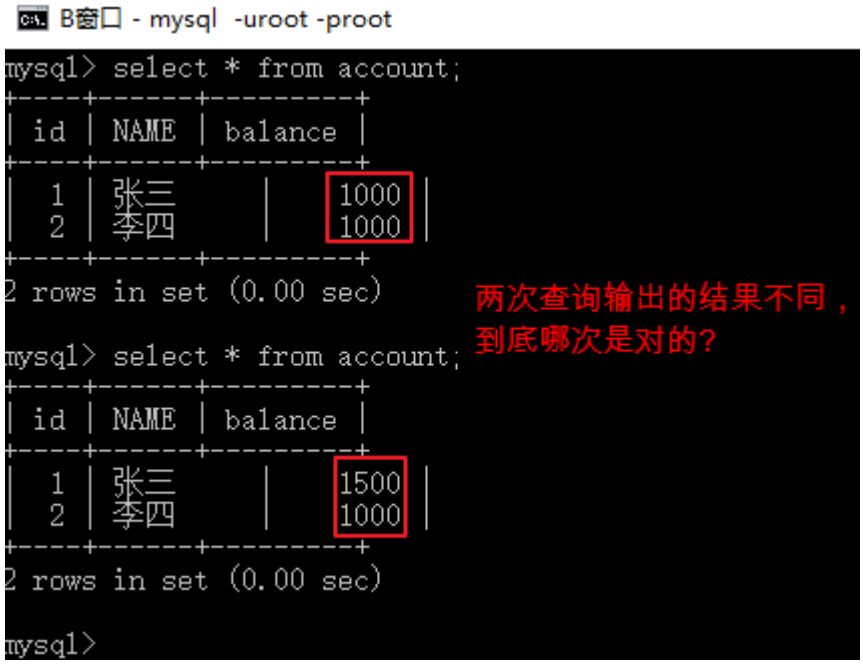
3. 在A窗口开启事务，并更新数据

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



4. B窗口查询

```
select * from account;
```

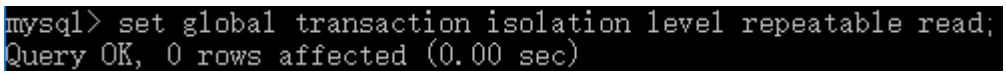


两次查询输出的结果不同，到底哪次是对的？不知道以哪次为准。 很多人认为这种情况就对了，无须困惑，当然是后面的为准。我们可以考虑这样一种情况，比如银行程序需要将查询结果分别输出到电脑屏幕和发短信给客户，结果在一个事务中针对不同的输出目的地进行的两次查询不一致，导致文件和屏幕中的结果不一致，银行工作人员就不知道以哪个为准了。

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;`

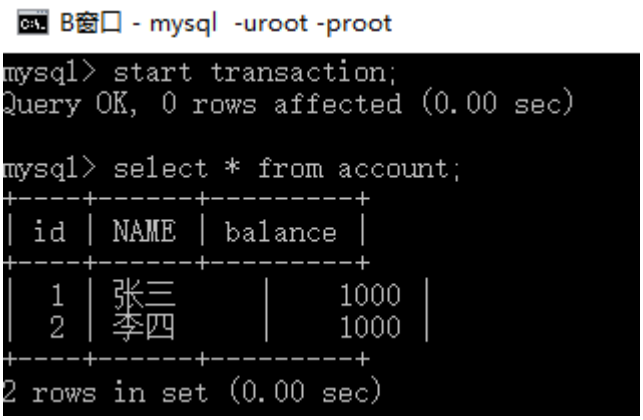
1. A窗口设置隔离级别为：`repeatable read`

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



2. B窗口退出MySQL，B窗口再进入MySQL

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



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

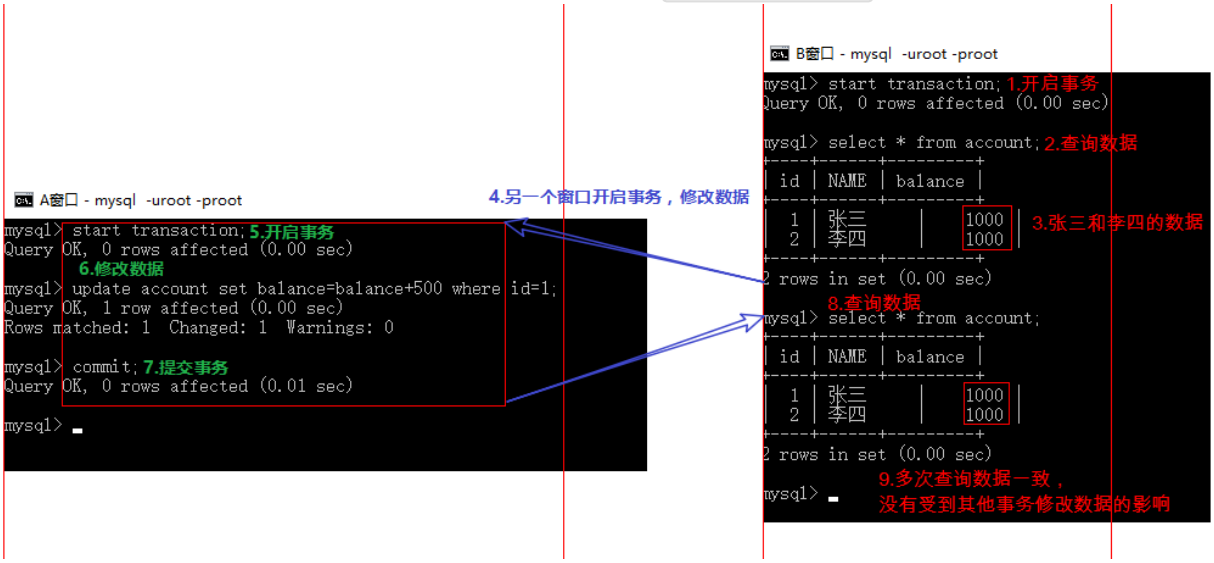
```
select * from account;
```

B窗口 - mysql -uroot -proot

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

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

B窗口查询了2次数据不变

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

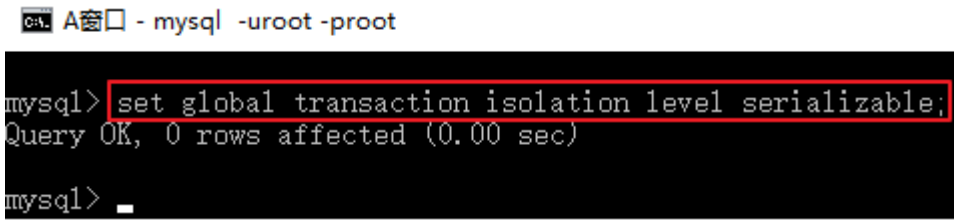


1.5.2.3 幻读的演示

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;`

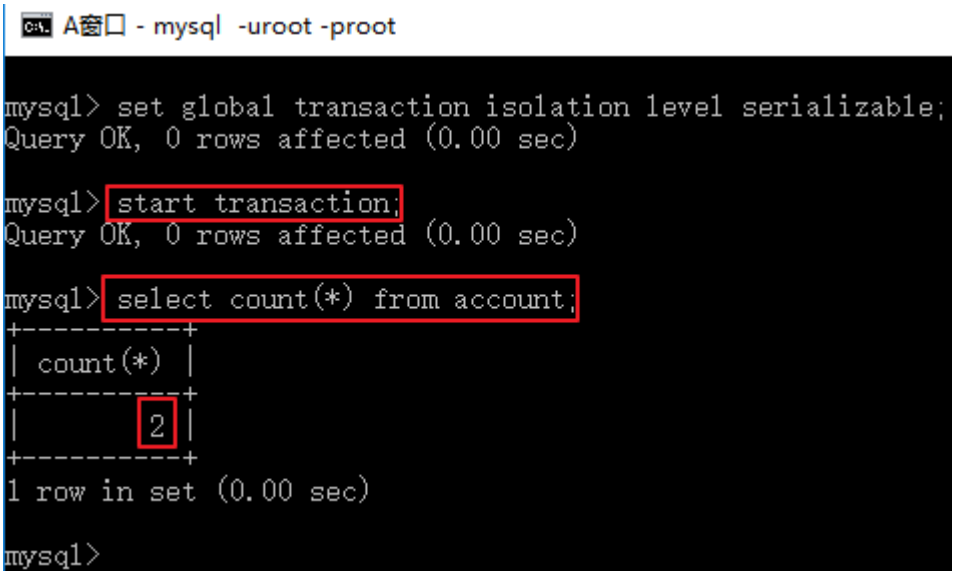
1. 开启A窗口

```
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;
```

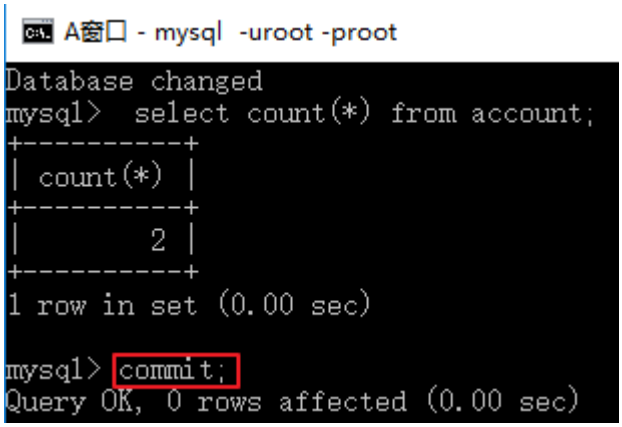


3. 再开启B窗口，登录MySQL

4. 在B窗口中开启事务，添加一条记录

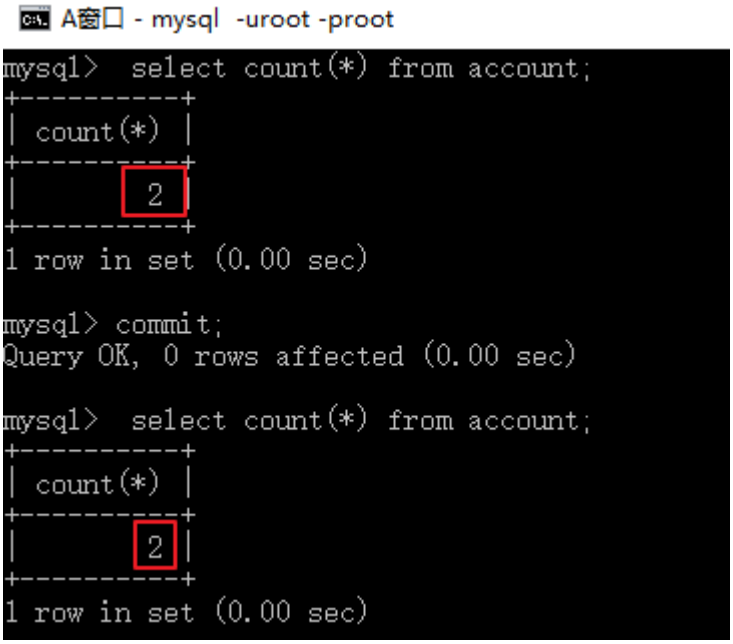
```
start transaction; -- 开启事务
insert into account (name,balance) values ('LaoWang', 500);
```

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

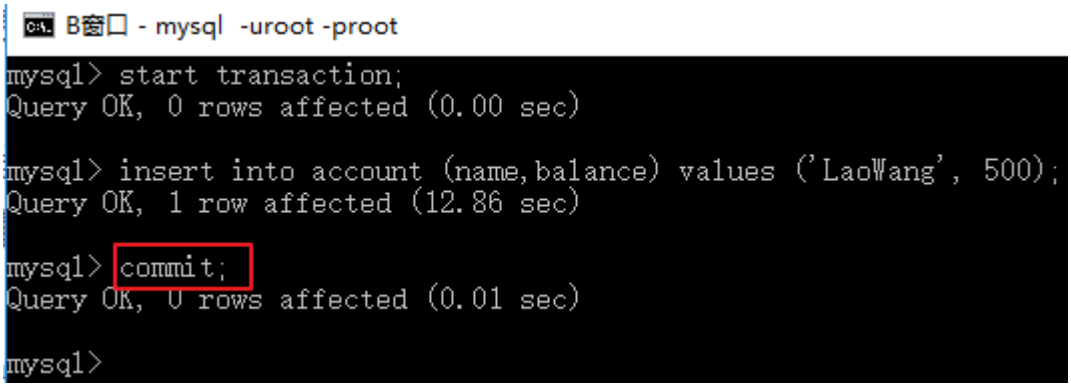


6. 在A窗口中接着查询，发现数据不变

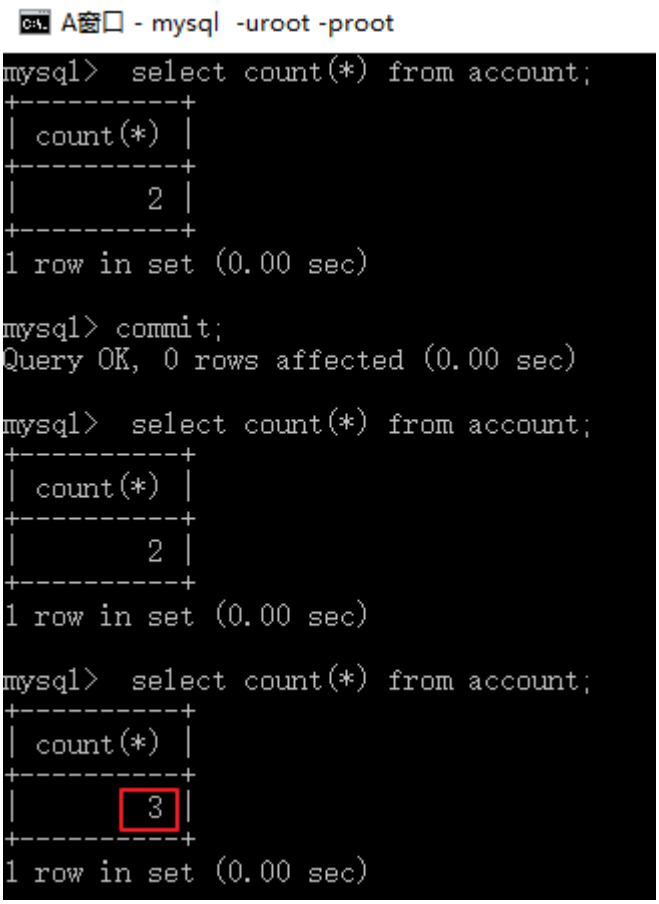
```
select count(*) from account;
```



7. B窗口中commit提交当前事务



8. A窗口就能看到最新的数据



> 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

第2章 多表查询

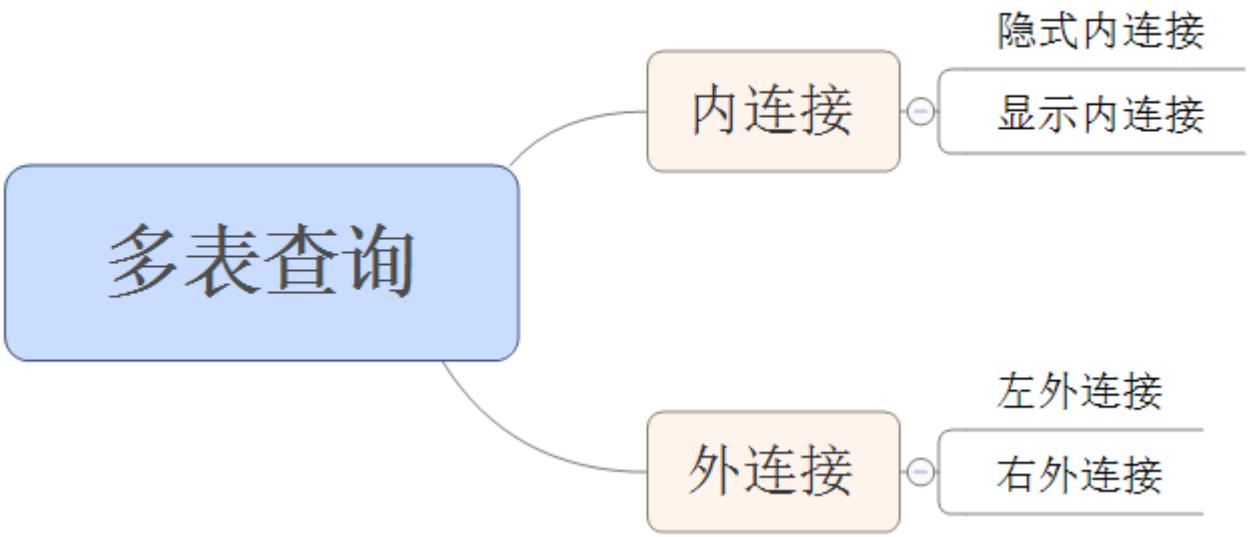
2.1 什么是多表查询

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	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

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，这种效果成为笛卡尔积

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乘积

左表的每条数据和右表的每条数据组合，这种效果成为笛卡尔乘积

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。

左表的每条数据和右表的每条数据组合，这种效果成为笛卡尔乘积

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

2.6 子查询

一条SELECT语句结果作为另一条SELECT语法一部分（查询条件，查询结果，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. 子查询结果是单例多行的时候

dept_id
1
2

3. 子查询的结果是多行多列

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` ，肯定在 `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`

2.6.1 子查询的结果是一个值的时候

子查询结果只要是 `单列` ，肯定在 `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？**
    1. 查询最高工资是多少

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

MAX(salary)
9000

    1. 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？**
    1. 查询平均工资是多少

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

avg(salary)
5994.333333333333

1. 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

2.6.2 子查询结果是单例多行的时候

子查询结果只要是`单列`，肯定在`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`运算符`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**
    1. 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)

    1. 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);

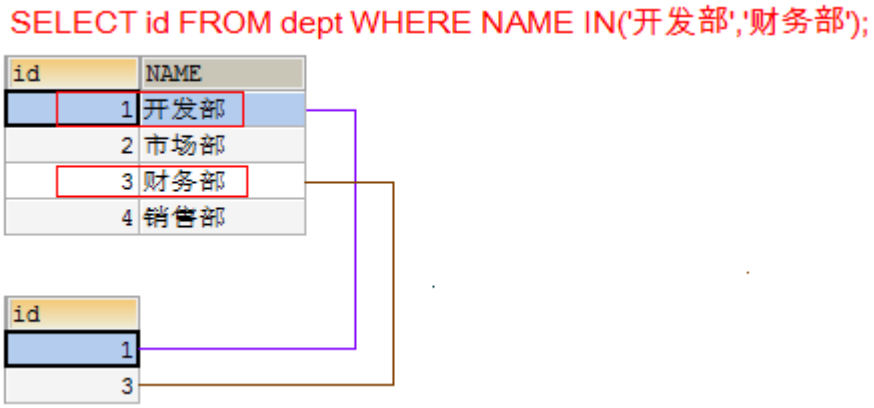
id	NAME
1	开发部
2	市场部
3	财务部
4	销售部

name
开发部
市场部

2. **查询开发部与财务部所有的员工信息**

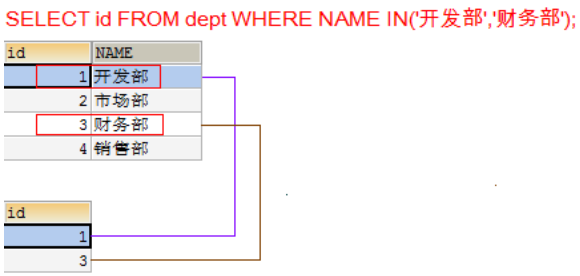
    1. 先查询开发部与财务部的id

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



    1. 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 * 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

2.6.3 子查询的结果是多行多列

子查询结果只要是`多列`，肯定在`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**

    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';
```

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)

1. 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';
```

2.6.4总结

> - 子查询结果只要是`单列`，肯定在`WHERE`后面作为`条件`
>   `SELECT 查询字段 FROM 表 WHERE 字段=（子查询）;`
> - 子查询结果只要是`多列`，肯定在`FROM`后面作为`表`
>   `SELECT 查询字段 FROM （子查询） 表别名 WHERE 条件;`