第三讲 表间关系与多表查询

一、表与表之间的关系

1. 实体关系

现实生活中，实体与实体之间肯定是有关系的，比如：部门和员工，老师和学生等。那么我们 在设计表的时候，就应该体现出这种关系！

实体之间的三种关系

- 一对多：最常用的关系 部门和员工
- 多对多：选课系统中 课程和学生的关系， 一门课程可以有多个学生选择，一个学生选择多门课程
- 一对一：相对使用比较少。HR 系统中，员工与简历

① 一对多 （录屏 19-1）

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,字段作为外键指向主表(一方)的主键

举例：

- 需求：一个旅游线路分类中有多个旅游线路
- 界面：

- 表与表的关系 ：

rid	rname	price	rdate	cid
1	【厦门+鼓浪屿+南普陀寺+曾厝垵 高铁3天 惠贵团】尝	1499	2018-01-27	1
2	【浪漫桂林 阳朔西街高铁3天纯玩 高级团】城徽象鼻山	699	2018-02-22	3
3	【爆款￥1699秒杀】泰国 曼谷 芭堤雅 金沙岛 杜拉拉水	1699	2018-01-27	2
4	【经典•狮航 ￥2399秒杀】巴厘岛双飞五天 抵玩【广州	2399	2017-12-23	2
5	香港迪士尼乐园自由行2天【永东跨境巴士广东至迪士尼	799	2018-04-10	4

cid	cname
1	周边游
2	出境游
3	国内游
4	港澳游

- 代码：

```
-- 创建旅游线路分类表 tab_category
-- cid 旅游线路分类主键，自动增长
-- cname 旅游线路分类名称非空，唯一，字符串最长 100
create table tab_category (
    cid int primary key auto_increment,
    cname varchar(100) not null unique
 )

-- 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');
 select * from tab_category;

-- 创建旅游线路表 tab_route
/*
rid 旅游线路主键，自动增长
rname 旅游线路名称非空，唯一，字符串 100
price 价格
rdate 上架时间，日期类型
cid 外键，所属分类
*/
create table tab_route(
```

```
    rid int primary key auto_increment,
    rname varchar(100) not null unique,
    price double,
    rdate date,
    cid int,
    foreign key (cid) references tab_category(cid)
)

-- 添加旅游线路数据
INSERT INTO tab_route VALUES (NULL, '【厦门+鼓浪屿+南普陀寺+曾厝垵 高铁 3 天惠贵团】尝味友鸭面线 住 1 晚鼓浪屿', 1499, '2018-01-27', 1), (NULL, '【浪漫桂林 阳朔西街高铁 3 天纯玩 高级团】城徽象鼻山 兴坪漓江 西山公园', 699, '2018-0222', 3), (NULL, '【爆款￥1699 秒杀】泰国 曼谷 芭堤雅 金沙岛 杜拉拉水上市场 双飞六天【含送签费 泰风情 广州 往返 特价团】', 1699, '2018-01-27', 2), (NULL, '【经典 • 狮航￥2399 秒杀】巴厘岛双飞五天 抵玩【广州往返 特价团】', 2399, '2017-12-23', 2), (NULL, '香港迪士尼乐园自由行 2 天【永东跨境巴士广东至迪士尼去程交通+迪士尼一日门票+香港如心海景酒店 暨会议中心标准房 1 晚住宿】', 799, '2018-04-10', 4);

select * from tab_route;
```

② 多对多 （录屏 19-2）

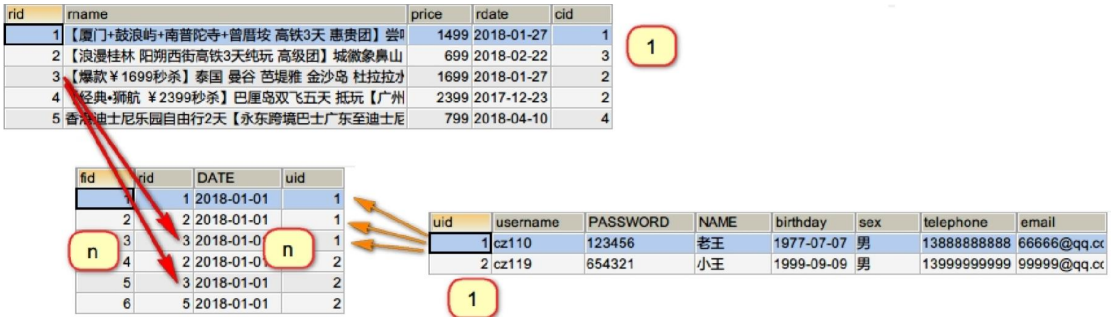
多对多（m:n） 例如：老师和学生，学生和课程，用户和角色

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。

举例：

- 需求：一个用户收藏多个线路，一个线路被多个用户收藏

- 表与表关系



- 代码

```
/*
创建用户表 tab_user
uid 用户主键，自增长
username 用户名长度 100，唯一，非空
password 密码长度 30，非空
name 真实姓名长度 100
birthday 生日
sex 性别，定长字符串 1
telephone 手机号，字符串 11
email 邮箱，字符串长度 100
*/
create table tab_user (
    uid int primary key auto_increment,
    username varchar(100) unique not null,
    password varchar(30) not null,
    name varchar(100),
    birthday date,
```

```
    sex char(1) default '男',
    telephone varchar(11),
    email varchar(100)
 )

-- 添加用户数据
INSERT INTO tab_user VALUES (NULL, 'cz110', 123456, '老王', '1977-07-07', '男',
'13888888888', '66666@qq.com'), (NULL, 'cz119', 654321, '小王', '1999-09-09', '男',
'13999999999', '99999@qq.com');
 select * from tab_user;

 /*  创建收藏表 tab_favorite
rid 旅游线路 id，外键
date 收藏时间
uid 用户 id，外键
rid 和 uid 不能重复，设置复合主键，同一个用户不能收藏同一个线路两次 */
create table tab_favorite (
    rid int,
    date datetime,
    uid int,
    -- 创建复合主键
    primary key(rid,uid),
    foreign key (rid) references tab_route(rid),
    foreign key(uid) references tab_user(uid)
 )

-- 增加收藏表数据
INSERT INTO tab_favorite VALUES (1, '2018-01-01', 1), -- 老王选择厦门
(2, '2018-02-11', 1), -- 老王选择桂林
(3, '2018-03-21', 1), -- 老王选择泰国
(2, '2018-04-21', 2), -- 小王选择桂林
(3, '2018-05-08', 2), -- 小王选择泰国
(5, '2018-06-02', 2); -- 小王选择迪士尼

 select * from tab_favorite;
```

③ 一对一

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：

一对一的建表原则	说明
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

二、 表连接查询

数据准备

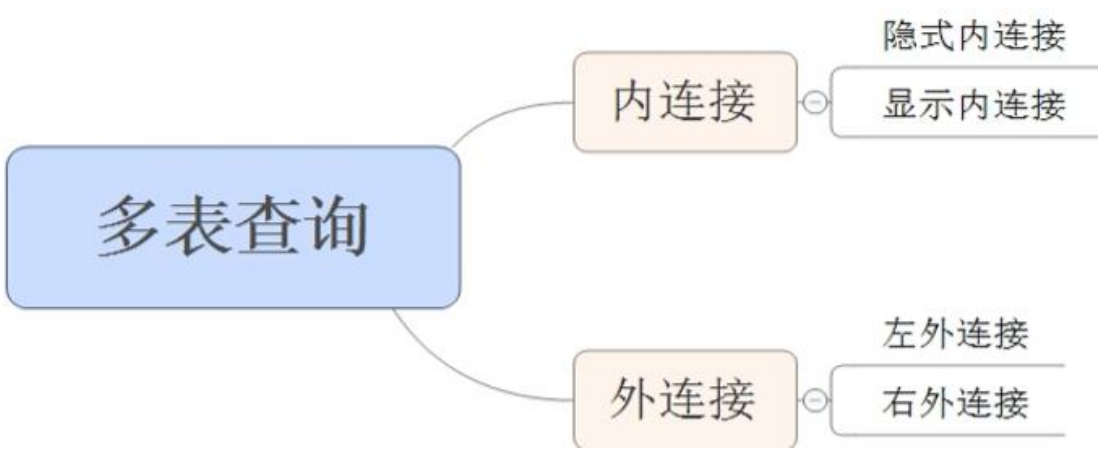
```
# 创建部门表
create table dept(
    id int primary key auto_increment,
    name varchar(20)
 )


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,
    foreign key (dept_id) references dept(id) -- 外键，关联部门表(部门表的主键)
 )

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,'200808-08',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);
```

1. 多表查询的分类（录屏 20）



2. 笛卡尔积现象

```
-- 需求：查询所有的员工和所有的部门
select * from emp,dept;
```

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，这种效果成为笛卡尔乘积

- 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。

```
-- 设置过滤条件 Column 'id' in where clause is ambiguous select * from emp,dept where id=5;
 select * from emp,dept where emp.`dept_id` = dept.`id`;

-- 查询员工和部门的名字 select emp.`name`, dept.`name` from emp,dept where emp.`dept_id` = dept.`id`;
```

3. 内连接 （录屏 21）

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。如：从表.外键=主表.主键

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

```
select * from emp,dept where emp.`dept_id` = dept.`id`;
```

② 显式内连接

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

③ 案例

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，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

- 确定查询哪些表

```
select * from emp inner join dept;
```

- 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 emp e inner join dept d on e.`dept_id` = d.`id`;
```

- 确定查询条件，我们查询的是唐僧的信息，员工表.name='唐僧'

```
select * from emp e inner join dept d on e.`dept_id` = d.`id` where e.`name`='唐僧 ';
```

- 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 e.`id`,e.`name`,e.`gender`,e.`salary`,d.`name` from emp e inner join dept d on e.`dept_id` = d.`id` where e.`name`='唐僧';
```

- 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` 编号,e.`name` 姓名,e.`gender` 性别,e.`salary` 工资,d.`name` 部门名字 from emp e inner join dept d on e.`dept_id` = d.`id` where e.`name`='唐僧';
```

④ 内连接查询步骤

- 确定查询哪些表
- 确定表连接的条件
- 确定查询的条件

- 确定查询的字段

4. 左外连接 （录屏 22）

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(左表是部门，右表员工)

举例：

```
-- 在部门表中增加一个销售部
insert into dept (name) values ('销售部'); select * from dept;

-- 使用内连接查询
select * from dept d inner join emp e on d.`id` = e.`dept_id`;

-- 使用左外连接查询
select * from dept d left join emp e on d.`id` = e.`dept_id`;
```

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)

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

举例：

```
-- 在员工表中增加一个员工
```

```
insert into emp values (null, '沙僧','男',6666,'2013-12-05',null);
select * from emp;
-- 使用内连接查询
select * from dept inner join emp on dept.`id` = emp.`dept_id`;

-- 使用右外连接查询
select * from dept right join emp on dept.`id` = emp.`dept_id`;
```

三、子查询 （录屏 23）

1. 子查询的概念

- 一个查询的结果做为另一个查询的条件
- 有查询的嵌套，内部的查询称为子查询
- 子查询要使用括号

举例：

```
-- 需求：查询开发部中有哪些员工
select * from emp;

-- 通过两条语句查询
select id from dept where name='开发部' ;
select * from emp where dept_id = 1;

-- 使用子查询  select * from emp where dept_id = (select id from dept where name='市场部');
```

2. 子查询结果的三种情况

① 子查询的结果是单行单列 (一个值)

子查询结果只要是单行单列，肯定在 WHERE 后面作为条件，父查询使用：比较运算符，如：> 、<、<>、 = 等

SELECT 查询字段 FROM 表 WHERE 字段=（子查询）;

案例：

- 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;
 -- 2) 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);
```

- 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;
-- 2) 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);
```

② 子查询的结果是多行单列

子查询结果是单列多行，结果集类似于一个数组，父查询使用 IN 运算符

举例：

- 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;
-- 再查询在这些部门 id 中部门的名字      Subquery returns more than 1 row
select name from dept where id = (select dept_id from emp where salary > 5000);
select name from dept where id in (select dept_id from emp where salary > 5000);
```

- 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');
 -- 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'));
```

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，否则这张表没有名称则无法访问表中的字段

举例：

- 查询出 2011 年以后入职的员工信息，包括部门名称

```
-- 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';

-- 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 d.`id`= e.dept_id ;
```

也可以使用表连接：

```
select * from emp inner join dept on emp.`dept_id` = dept.`id` where join_date >='2011-1-1';
select * from emp inner join dept on emp.`dept_id` = dept.`id` and join_date >='2011-1-1'; 4
```

四、事务

1. 什么是事务 录屏 24

在实际的开发过程中，一个业务操作如：转账，往往是要多次访问数据库才能完成的。转账是一个用户扣钱，另一个用户加钱。如果其中有一条 SQL 语句出现异常，这条 SQL 就

可能执行失败。

事务执行是一个整体，所有的 SQL 语句都必须执行成功。如果其中有 1 条 SQL 语句出现异常，则所有的 SQL 语句都要回滚，整个业务执行失败。

举例：转账的操作

```
-- 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： 张三账号-500 李四账号+500

```
-- 张三账号-500
update account set balance = balance - 500 where name='张三';
-- 李四账号+500
update account set balance = balance + 500 where name='李四';
```

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。

2. 手动提交事务

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

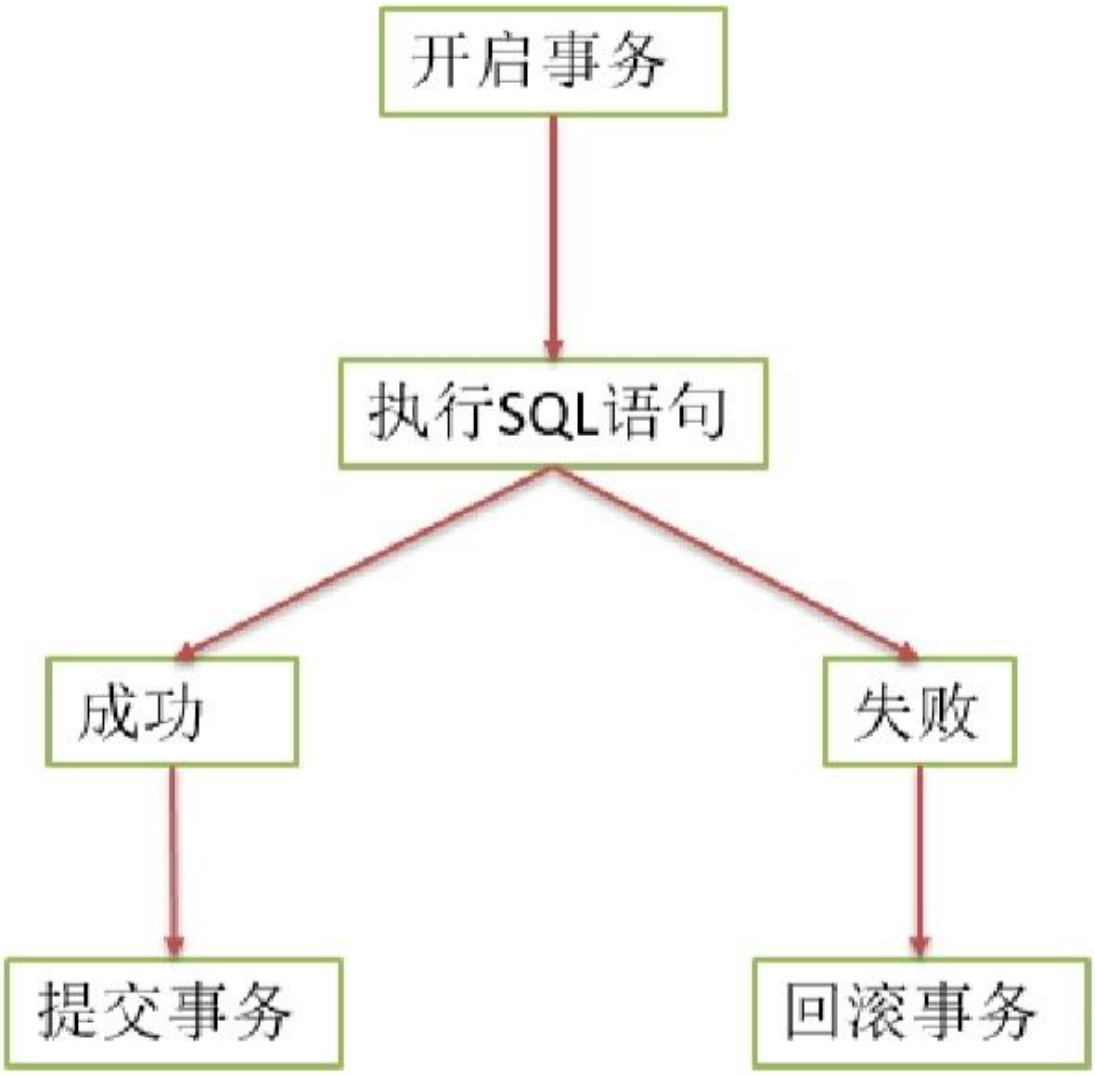
① 手动提交事务的 SQL 语句

功能	SQL 语句
开启事务	start transaction;
提交事务	commit;

回滚事务	rollback;

② 手动提交事务使用过程

- 执行成功的情况： 开启事务 -> 执行多条 SQL 语句 -> 成功提交事务
- 执行失败的情况： 开启事务 -> 执行多条 SQL 语句 -> 事务的回滚



3. 案例演示 录屏 25

① 事务提交

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

- 使用 DOS 控制台进入 MySQL
- 执行以下 SQL 语句： 1.开启事务， 2.张三账号-500， 3.李四账号+500
- 使用 SQLYog 查看数据库：发现数据并没有改变
- 在控制台执行 commit 提交事务：
- 使用 SQLYog 查看数据库：发现数据改变

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

mysql> update account set balance = balance - 500 where name='张三';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set balance = balance + 500 where name='李四';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

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

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

② 事务回滚

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	1000
2	李四	1000

- 在控制台执行以下 SQL 语句：1.开启事务， 2.张三账号-500
- 使用 SQLYog 查看数据库：发现数据并没有改变
- 在控制台执行 rollback 回滚事务：
- 使用 SQLYog 查看数据库：发现数据没有改变

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

mysql> update account set balance = balance - 500 where name='张三';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set balance = balance + 500 where name='李四';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

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

mysql> rollback;
```

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。

4. 自动提交事务

MySQL 默认每一条 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，语句执行完毕 自动提交事务，MySQL 默认开始自动提交事务

① 案例演示：自动提交事务

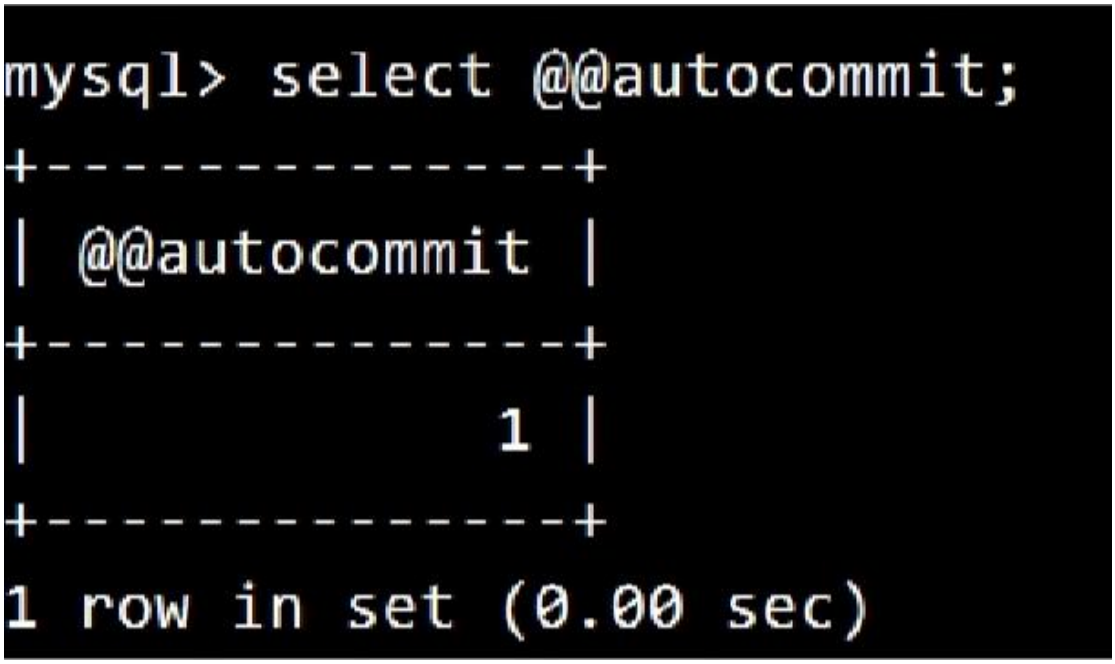
- 将金额重置为 1000
- 更新其中某一个账户
- 使用 SQLYog 查看数据库：发现数据已经改

```
mysql> update account set balance = balance + 500 where name='李四';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+----+------+---------+
| id | NAME | balance |
+----+------+---------+
|  1 | 张三     |    1000 |
|  2 | 李四     |    1500 |
+----+------+---------+
```

② 取消自动提交

- 查看 MySQL 是否开启自动提交事务



@@表示全局变量，1 表示开启，0 表示关闭

- 取消自动提交事务

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

mysql> select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
1 row in set (0.00 sec)
```

执行更新语句，使用 SQLYog 查看数据库，发现数据并没有改变

在控制台执行 commit 提交任务

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

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

5. 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中，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)

事务的步骤：

1) 客户端连接数据库服务器，创建连接时创建此用户临时日志文件

2) 开启事务以后，所有的操作都会先写入到临时日志文件中

3) 所有的查询操作从表中查询，但会经过日志文件加工后才返回

4) 如果事务提交则将日志文件中的数据写到表中，否则清空日志文件。

6. 事务的隔离级别

① 事务的四大特性 ACID

事务特性	含义
原子性（Atomicity）	每个事务都是一个整体，不可再拆分，事务中所有的 SQL 语句要么都执行成功， 要么都失败。
一致性（Consistency）	事务在执行前数据库的状态与执行后数据库的状态保持一致。如：转账前 2 个人的 总金额是 2000，转账后 2 个人总金额也是 2000
隔离性（Isolation）	事务与事务之间不应该相互影响，执行时保持隔离的状态
持久性（Durability）	一旦事务执行成功，对数据库的修改是持久的。就算关机，也是保存下来的

② 事务的隔离级别

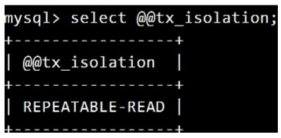
事务在操作时的理想状态： 所有的事务之间保持隔离，互不影响。因为并发操作，多个用户同时访问同一个数据。可能引发并发访问的问题：

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

③ 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	

隔离级别越高，性能越差，安全性越高

查询全局事务隔离级别 查询隔离级别 **select @@tx_isolation;**