mysql高级

今日目标

> - 掌握约束的使用
> - 掌握表关系及建表原则
> - 重点掌握多表查询操作
> - 掌握事务操作

1，约束

id	name	age	sex	address	math	english	hire_date
1	马运	55	男	杭州	-5	78	1995-09-01
(Null)	马花疼	45	女	深圳	98	87	1998-09-01
1	马斯克	55	男	香港	56	77	1999-09-02
1	柳白	3000	女	湖南	76	65	1997-09-05
5	柳青	20	男	湖南	86	(Null)	1998-09-01
6	刘德花	57	男	香港	99	99	1998-09-01
7	张学右	22	女	香港	99	99	1998-09-01
8	德玛西亚	18	男	南京	56	65	1994-09-02

上面表中可以看到表中数据存在一些问题：

- id 列一般是用标示数据的唯一性的，而上述表中的id为1的有三条数据，并且 `马花疼` 没有id进行标示
- `柳白` 这条数据的age列的数据是3000，而人也不可能活到3000岁
- `马运` 这条数据的math数学成绩是-5，而数学学得再不好也不可能出现负分
- `柳青` 这条数据的english列（英文成绩）值为null，而成绩即使没考也得是0分

针对上述数据问题，我们就可以从数据库层面在添加数据的时候进行限制，这个就是约束。

1.1 概念

- 约束是作用于表中列上的规则，用于限制加入表的数据

  例如：我们可以给id列加约束，让其值不能重复，不能为null值。

- 约束的存在保证了数据库中数据的正确性、有效性和完整性

  添加约束可以在添加数据的时候就限制不正确的数据，年龄是3000，数学成绩是-5分这样无效的数据，继而保障数据的完整性。

1.2 分类

- **非空约束： 关键字是 NOT NULL**

  保证列中所有的数据不能有null值。

  例如：id列在添加 `马花疼` 这条数据时就不能添加成功。

- **唯一约束：关键字是 UNIQUE**

  保证列中所有数据各不相同。

  例如：id列中三条数据的值都是1，这样的数据在添加时是绝对不允许的。

- **主键约束： 关键字是 PRIMARY KEY**

  主键是一行数据的唯一标识，要求非空且唯一。一般我们都会给没张表添加一个主键列用来唯一标识数据。

  例如：上图表中id就可以作为主键，来标识每条数据。那么这样就要求数据中id的值不能重复，不能为null值。

- **检查约束： 关键字是 CHECK**

  保证列中的值满足某一条件。

  例如：我们可以给age列添加一个范围，最低年龄可以设置为1，最大年龄就可以设置为300，这样的数据才更合理些。

> 注意：MySQL不支持检查约束。

> 这样是不是就没办法保证年龄在指定的范围内了？从数据库层面不能保证，以后可以在java代码中进行限制，一样也可以实现要求。

- **默认约束： 关键字是 DEFAULT**

  保存数据时，未指定值则采用默认值。

  例如：我们在给english列添加该约束，指定默认值是0，这样在添加数据时没有指定具体值时就会采用默认给定的0。

- **外键约束： 关键字是 FOREIGN KEY**

  外键用来让两个表的数据之间建立链接，保证数据的一致性和完整性。

  外键约束现在可能还不太好理解，后面我们会重点进行讲解。

1.3 非空约束

- 概念

  非空约束用于保证列中所有数据不能有NULL值

- 语法

  - 添加约束

```
-- 创建表时添加非空约束
CREATE TABLE 表名(
    列名 数据类型 NOT NULL,
    …
);

```

```
-- 建完表后添加非空约束
ALTER TABLE 表名 MODIFY 字段名 数据类型 NOT NULL;
```

  - 删除约束

```
ALTER TABLE 表名 MODIFY 字段名 数据类型;
```

1.4 唯一约束

- 概念

  唯一约束用于保证列中所有数据各不相同

- 语法

  - 添加约束

```
-- 创建表时添加唯一约束
CREATE TABLE 表名(
    列名 数据类型 UNIQUE [AUTO_INCREMENT],
    -- AUTO_INCREMENT: 当不指定值时自动增长
    …
);
CREATE TABLE 表名(
    列名 数据类型,
    …
    [CONSTRAINT] [约束名称] UNIQUE(列名)
);
```

```
-- 建完表后添加唯一约束
ALTER TABLE 表名 MODIFY 字段名 数据类型 UNIQUE;
```

  - 删除约束

```
ALTER TABLE 表名 DROP INDEX 字段名;
```

1.5 主键约束

- 概念

  主键是一行数据的唯一标识，要求非空且唯一

  一张表只能有一个主键

- 语法
  - 添加约束

```sql
-- 创建表时添加主键约束
CREATE TABLE 表名(
    列名 数据类型 PRIMARY KEY [AUTO_INCREMENT],
    …
);
CREATE TABLE 表名(
    列名 数据类型,
    [CONSTRAINT] [约束名称] PRIMARY KEY(列名)
);

```

```sql
-- 建完表后添加主键约束
ALTER TABLE 表名 ADD PRIMARY KEY(字段名);
```

  - 删除约束

```sql
ALTER TABLE 表名 DROP PRIMARY KEY;
```

1.6 默认约束

- 概念

  保存数据时，未指定值则采用默认值

- 语法
  - 添加约束

```sql
-- 创建表时添加默认约束
CREATE TABLE 表名(
    列名 数据类型 DEFAULT 默认值,
    …
);
```

```sql
-- 建完表后添加默认约束
ALTER TABLE 表名 ALTER 列名 SET DEFAULT 默认值;
```

  - 删除约束

```sql
ALTER TABLE 表名 ALTER 列名 DROP DEFAULT;
```

1.7 约束练习

根据需求，为表添加合适的约束

```sql
-- 员工表
CREATE TABLE emp (
    id INT,  -- 员工id，主键且自增长
    ename VARCHAR(50), -- 员工姓名，非空且唯一
    joindate DATE,  -- 入职日期，非空
    salary DOUBLE(7,2),  -- 工资，非空
    bonus DOUBLE(7,2)  -- 奖金，如果没有将近默认为0
);
```

上面一定给出了具体的要求，我们可以根据要求创建这张表，并为每一列添加对应的约束。建表语句如下：

```sql
DROP TABLE IF EXISTS emp;

-- 员工表
CREATE TABLE emp (
  id INT PRIMARY KEY, -- 员工id，主键且自增长
  ename VARCHAR(50) NOT NULL UNIQUE, -- 员工姓名，非空并且唯一
  joindate DATE NOT NULL , -- 入职日期，非空
  salary DOUBLE(7,2) NOT NULL , -- 工资，非空
  bonus DOUBLE(7,2) DEFAULT 0 -- 奖金，如果没有奖金默认为0
);
```

通过上面语句可以创建带有约束的 `emp` 表，约束能不能发挥作用呢。接下来我们一一进行验证，先添加一条没有问题的数据

```sql
INSERT INTO emp(id,ename,joindate,salary,bonus) values(1,'张三','1999-11-11',8800,5000);
```

- **验证主键约束，非空且唯一**

```sql
INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'张三','1999-11-11',8800,5000);
```

执行结果如下：

信息 概况 状态

[SQL]-- 演示主键约束：非空且唯一
INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'张三','1999-11-11',8800,5000);
[Err] 1048 - Column 'id' cannot be null

从上面的结果可以看到，字段 `id` 不能为null。那我们重新添加一条数据，如下：

```sql
INSERT INTO emp(id,ename,joindate,salary,bonus) values(1,'张三','1999-11-11',8800,5000);
```

执行结果如下：

信息 概况 状态

[SQL]INSERT INTO emp(id,ename,joindate,salary,bonus) values(1,'张三','1999-11-11',8800,5000);
[Err] 1062 - Duplicate entry '1' for key 'PRIMARY'

从上面结果可以看到，1这个值重复了。所以主键约束是用来限制数据非空且唯一的。那我们再添加一条符合要求的数据

```sql
INSERT INTO emp(id,ename,joindate,salary,bonus) values(2,'李四','1999-11-11',8800,5000);
```

执行结果如下：

信息 概况 状态

[SQL]INSERT INTO emp(id,ename,joindate,salary,bonus) values(2,'李四','1999-11-11',8800,5000);
受影响的行: 1
时间: 0.009s

- **验证非空约束**

```sql
INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,null,'1999-11-11',8800,5000);
```

执行结果如下：

信息 概况 状态

[SQL]-- 演示非空约束
INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,null,'1999-11-11',8800,5000);
[Err] 1048 - Column 'ename' cannot be null

从上面结果可以看到，`ename` 字段的非空约束生效了。

- **验证唯一约束**

```sql
INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,'李四','1999-11-11',8800,5000);
```

执行结果如下：

信息 概况 状态

[SQL]-- 演示唯一约束
INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,'李四','1999-11-11',8800,5000);
[Err] 1062 - Duplicate entry '李四' for key 'ename'

从上面结果可以看到，`ename` 字段的唯一约束生效了。

- **验证默认约束**

```
INSERT INTO emp(id,ename,joindate,salary) values(3,'王五','1999-11-11',8800);
```

执行完上面语句后查询表中数据，如下图可以看到王五这条数据的bonus列就有了默认值0。

信息 | 结果1 | 概况 | 状态

id	ename	joindate	salary	bonus
1	张三	1999-11-11	8800	5000
2	李四	1999-11-11	8800	5000
3	王五	1999-11-11	8800	0

注意：默认约束只有在不给值时才会采用默认值。如果给了null，那值就是null值。

如下：

```
INSERT INTO emp(id,ename,joindate,salary,bonus) values(4,'赵六','1999-11-11',8800,null);
```

执行完上面语句后查询表中数据，如下图可以看到赵六这条数据的bonus列的值是null。

信息 | 结果1 | 概况 | 状态

id	ename	joindate	salary	bonus
1	张三	1999-11-11	8800	5000
2	李四	1999-11-11	8800	5000
3	王五	1999-11-11	8800	0
4	赵六	1999-11-11	8800	(Null)

- **验证自动增长： auto_increment 当列是数字类型 并且唯一约束**

重新创建 `emp` 表，并给id列添加自动增长

```
-- 员工表
CREATE TABLE emp (
  id INT PRIMARY KEY auto_increment, -- 员工id，主键且自增长
  ename VARCHAR(50) NOT NULL UNIQUE, -- 员工姓名，非空并且唯一
  joindate DATE NOT NULL , -- 入职日期，非空
  salary DOUBLE(7,2) NOT NULL , -- 工资，非空
  bonus DOUBLE(7,2) DEFAULT 0 -- 奖金，如果没有奖金默认为0
);
```

接下来给emp添加数据，分别验证不给id列添加值以及给id列添加null值，id列的值会不会自动增长：

```
INSERT INTO emp(ename,joindate,salary,bonus) values('赵六','1999-11-11',8800,null);
INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'赵六2','1999-11-11',8800,null);
INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'赵六3','1999-11-11',8800,null);
```

1.8 外键约束

1.8.1 概述

外键用来让两个表的数据之间建立链接，保证数据的一致性和完整性。

如何理解上面的概念呢？如下图有两张表，员工表和部门表：

emp 员工表

id	name	age	dep_id
1	张三	20	1
2	李四	20	1
3	王五	20	1
4	赵六	20	2
5	孙七	22	2
6	周八	18	2

dept 部门表

id	dep_name	addr
1	研发部	广州
2	销售部	深圳

员工表中的dep_id字段是部门表的id字段关联，也就是说1号学生张三属于1号部门研发部的员工。现在我要删除1号部门，就会出现错误的数据（员工表中属于1号部门的数据）。而我们上面说的两张表的关系只是我们认为它们有关系，此时需要通过外键让这两张表产生数据库层面的关系，这样你要删除部门表中的1号部门的数据将无法删除。

1.8.2 语法

- 添加外键约束

```sql
-- 创建表时添加外键约束
CREATE TABLE 表名(
    列名 数据类型,
    …
    [CONSTRAINT] [外键名称] FOREIGN KEY(外键列名) REFERENCES 主表(主表列名)
);
```

```sql
-- 建完表后添加外键约束
ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段名称) REFERENCES 主表名称(主表列名称);
```

- 删除外键约束

```sql
ALTER TABLE 表名 DROP FOREIGN KEY 外键名称;
```

1.8.3 练习

根据上述语法创建员工表和部门表，并添加上外键约束：

```sql
-- 删除表
DROP TABLE IF EXISTS emp;
DROP TABLE IF EXISTS dept;

-- 部门表
CREATE TABLE dept(
    id int primary key auto_increment,
    dep_name varchar(20),
    addr varchar(20)
);
-- 员工表
CREATE TABLE emp(
    id int primary key auto_increment,
    name varchar(20),
    age int,
    dep_id int,

    -- 添加外键 dep_id,关联 dept 表的id主键
    CONSTRAINT fk_emp_dept FOREIGN KEY(dep_id) REFERENCES dept(id)
);
```

添加数据

```sql
-- 添加 2 个部门
insert into dept(dep_name,addr) values
('研发部','广州'),('销售部', '深圳');

-- 添加员工,dep_id 表示员工所在的部门
INSERT INTO emp (NAME, age, dep_id) VALUES
('张三', 20, 1),
('李四', 20, 1),
('王五', 20, 1),
('赵六', 20, 2),
('孙七', 22, 2),
('周八', 18, 2);
```

此时删除 `研发部` 这条数据，会发现无法删除。

删除外键

```sql
alter table emp drop FOREIGN key fk_emp_dept;
```

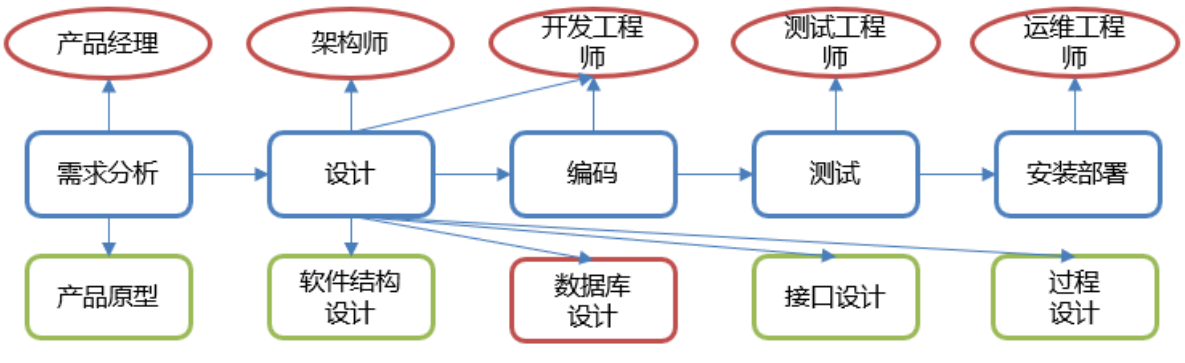
重新添加外键

```
alter table emp add CONSTRAINT fk_emp_dept FOREIGN key(dep_id) REFERENCES dept(id);
```

2，数据库设计

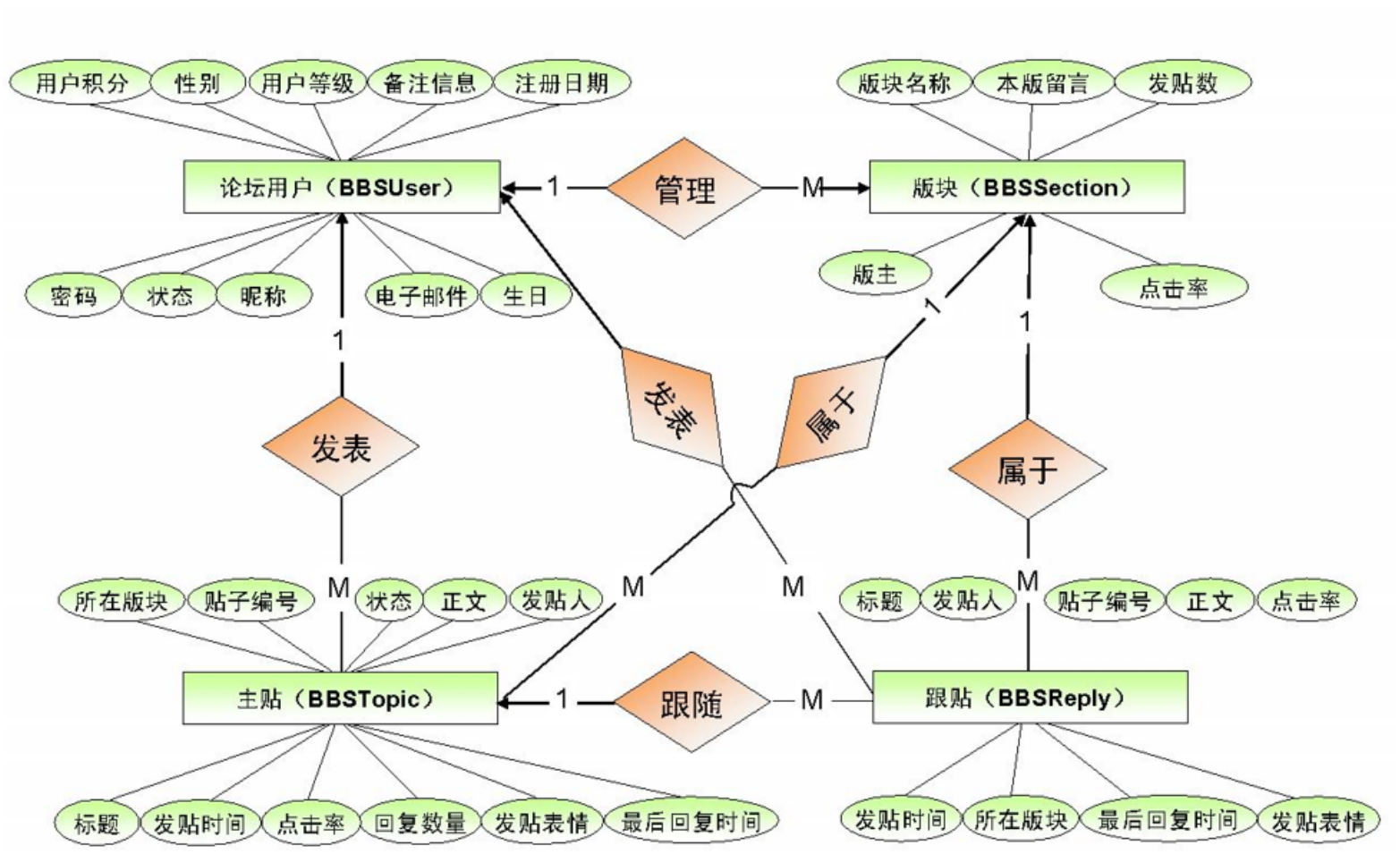
2.1 数据库设计简介

- 软件的研发步骤

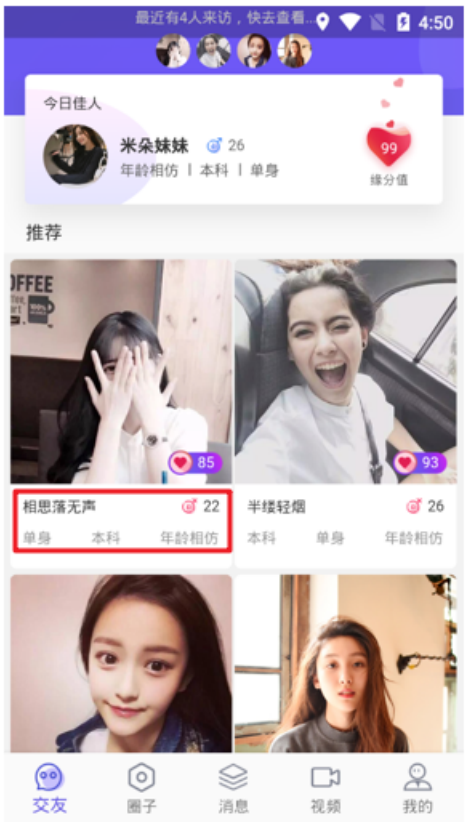


- 数据库设计概念
  - 数据库设计就是根据业务系统的具体需求，结合我们所选用的DBMS，为这个业务系统构造出最优的数据存储模型。
  - 建立数据库中的**表结构**以及**表与表之间的关联关系**的过程。
  - 有哪些表？表里有哪些字段？表和表之间有什么关系？
- 数据库设计的步骤
  - 需求分析（数据是什么? 数据具有哪些属性? 数据与属性的特点是什么）
  - 逻辑分析（通过ER图对数据库进行逻辑建模，不需要考虑我们所选用的数据库管理系统）

    如下图就是ER(Entity/Relation)图：



  - 物理设计（根据数据库自身的特点把逻辑设计转换为物理设计）
  - 维护设计（1.对新的需求进行建表；2.表优化）
- 表关系
  - 一对一
    - 如：用户 和 用户详情
    - 一对一关系多用于表拆分，将一个实体中经常使用的字段放一张表，不经常使用的字段放另一张表，用于提升查询性能



上图左边是用户的详细信息，而我们真正在展示用户信息时最长用的则是上图右边红框所示，所以我们会将详细信息查分成两周那个表。

- 一对多
  - 如：部门 和 员工
  - 一个部门对应多个员工，一个员工对应一个部门。如下图：

emp 员工表

id	name	age	dep_id
1	张三	20	1
2	李四	20	1
3	王五	20	1
4	赵六	20	2
5	孙七	22	2
6	周八	18	2

dept 部门表

id	dep_name	addr
1	研发部	广州
2	销售部	深圳

- 多对多
  - 如：商品 和 订单
  - 一个商品对应多个订单，一个订单包含多个商品。如下图：

2.2 表关系(一对多)

- 一对多
  - 如：部门 和 员工
  - 一个部门对应多个员工，一个员工对应一个部门。
- 实现方式

  在多的一方建立外键，指向一的一方的主键
- 案例

  我们还是以 `员工表` 和 `部门表` 举例:

tb_emp 员工表 **M**

id	name	age
1	张三	23
2	李四	24
3	王五	25

1 tb_dept 部门表

id	name	addr
1	财务部	北京
2	市场部	上海
3	研发部	成都

经过分析发现，员工表属于多的一方，而部门表属于一的一方，此时我们会在员工表中添加一列（dep_id），指向于部门表的主键（id）：

tb_emp 员工表 **M**

id	name	age	dep_id
1	张三	23	1
2	李四	24	1
3	王五	25	2

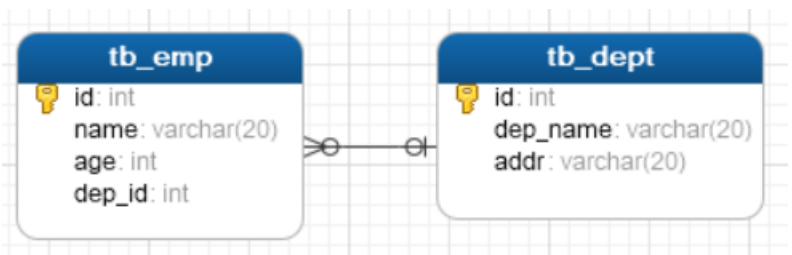
1 tb_dept 部门表

id	name	addr
1	财务部	北京
2	市场部	上海
3	研发部	成都

建表语句如下：

```sql
-- 删除表
DROP TABLE IF EXISTS tb_emp;
DROP TABLE IF EXISTS tb_dept;

-- 部门表
CREATE TABLE tb_dept(
    id int primary key auto_increment,
    dep_name varchar(20),
    addr varchar(20)
);
-- 员工表
CREATE TABLE tb_emp(
    id int primary key auto_increment,
    name varchar(20),
    age int,
    dep_id int,

    -- 添加外键 dep_id,关联 dept 表的id主键
    CONSTRAINT fk_emp_dept FOREIGN KEY(dep_id) REFERENCES tb_dept(id)
);
```

查看表结构模型图：



2.3 表关系(多对多)

- 多对多
  - 如：商品 和 订单
  - 一个商品对应多个订单，一个订单包含多个商品
- 实现方式

  建立第三张中间表，中间表至少包含两个外键，分别关联两方主键
- 案例

  我们以 `订单表` 和 `商品表` 举例：

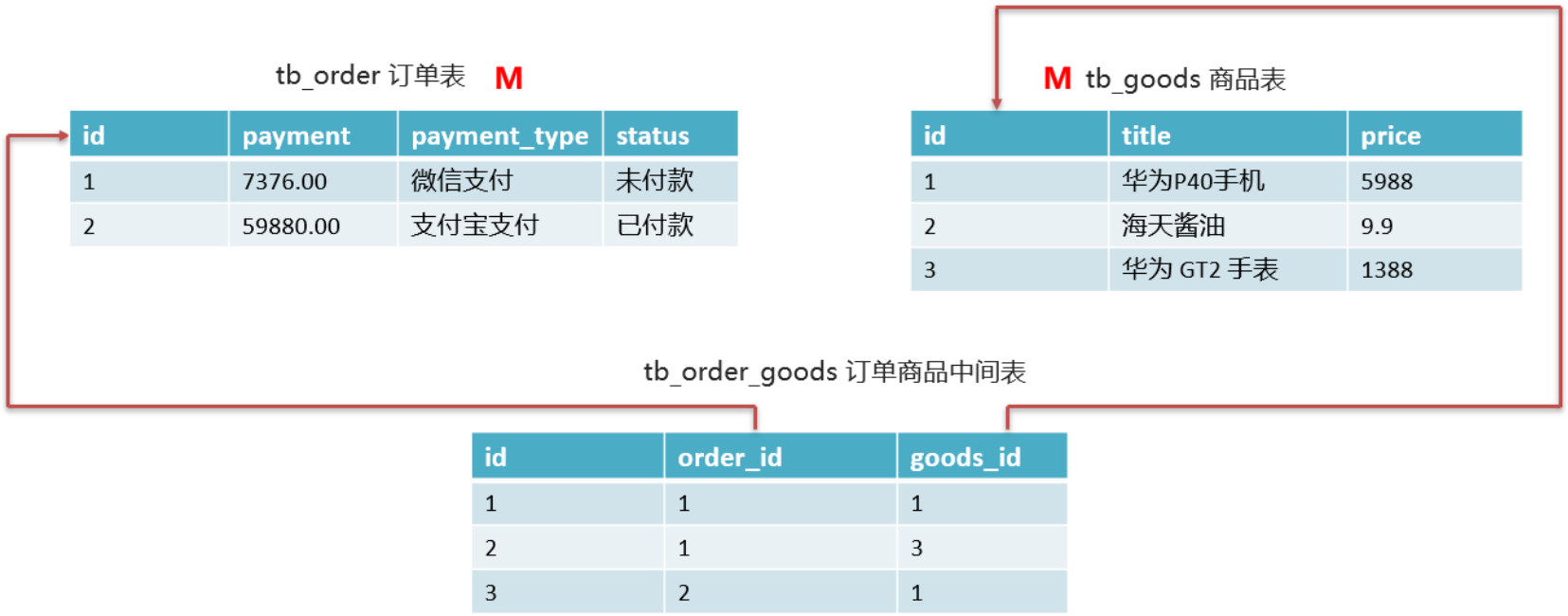
tb_order 订单表 M

id	payment	payment_type	status
1	7376.00	微信支付	未付款
2	59880.00	支付宝支付	已付款

M tb_goods 商品表

id	title	price
1	华为P40手机	5988
2	海天酱油	9.9
3	华为 GT2 手表	1388

经过分析发现，订单表和商品表都属于多的一方，此时需要创建一个中间表，在中间表中添加订单表的外键和商品表的外键指向两张表的主键：

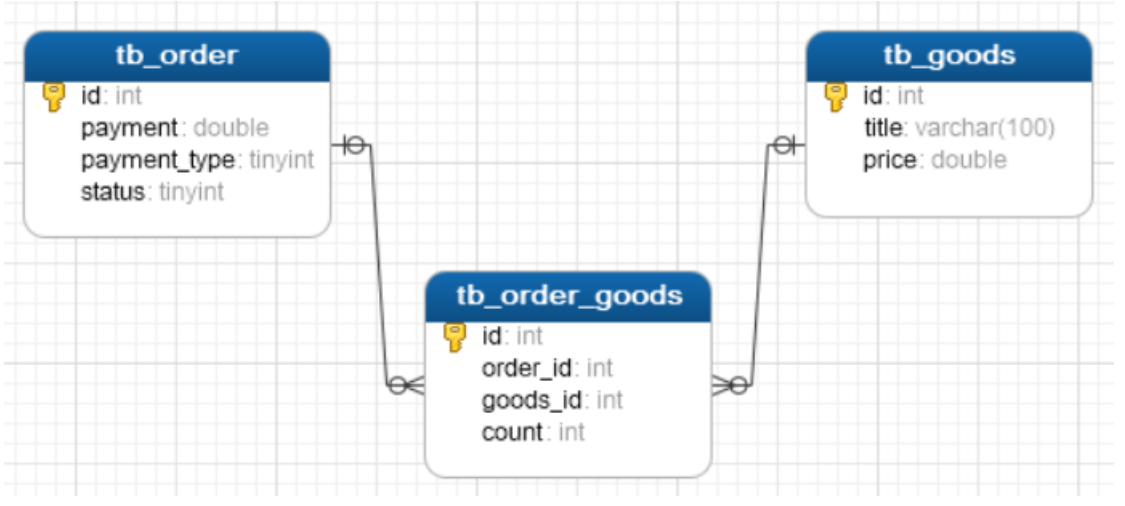


建表语句如下：

```sql
-- 删除表
DROP TABLE IF EXISTS tb_order_goods;
DROP TABLE IF EXISTS tb_order;
DROP TABLE IF EXISTS tb_goods;

-- 订单表
CREATE TABLE tb_order(
    id int primary key auto_increment,
    payment double(10,2),
    payment_type TINYINT,
    status TINYINT
);

-- 商品表
CREATE TABLE tb_goods(
    id int primary key auto_increment,
    title varchar(100),
    price double(10,2)
);

-- 订单商品中间表
CREATE TABLE tb_order_goods(
    id int primary key auto_increment,
    order_id int,
    goods_id int,
    count int
);
```

```
28
29  -- 建完表后，添加外键
30  alter table tb_order_goods add CONSTRAINT fk_order_id FOREIGN key(order_id) REFERENCES
    tb_order(id);
31  alter table tb_order_goods add CONSTRAINT fk_goods_id FOREIGN key(goods_id) REFERENCES
    tb_goods(id);
```

查看表结构模型图：



2.4 表关系(一对一)

- 一对一
  - 如：用户 和 用户详情
  - 一对一关系多用于表拆分，将一个实体中经常使用的字段放一张表，不经常使用的字段放另一张表，用于提升查询性能
- 实现方式

  在任意一方加入外键，关联另一方主键，并且设置外键为唯一(UNIQUE)
- 案例

  我们以 `用户表` 举例：

tb_user 用户表

id	photo	nickname	age	gender	city	edu	income	status	desc
1	a.jpg	一场梦	23	女	广州	硕士	3000	单身	...
2	b.png	风清扬	35	男	湖北	本科	30000	离异	...
3	c.jpg	赵云	41	男	河南	本科	40000	单身	...

而在真正使用过程中发现 id、photo、nickname、age、gender 字段比较常用，此时就可以将这张表查分成两张表。

tb_user 用户表

id	photo	nickname	age	gender	desc_id
1	a.jpg	一场梦	23	女	1
2	b.png	风清扬	35	男	2
3	c.jpg	赵云	41	男	3

1 → 1

tb_user_desc 用户详情表

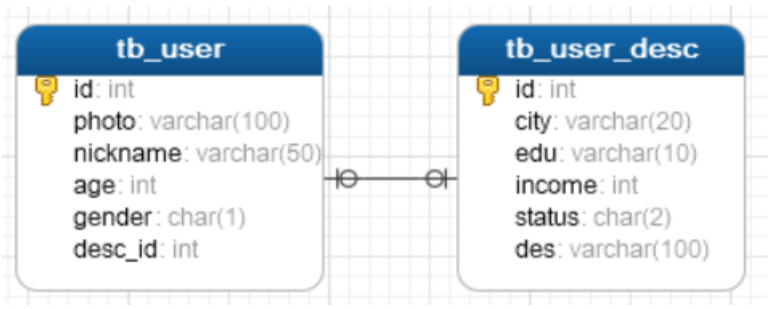
id	city	edu	income	status	desc
1	广州	硕士	3000	单身	...
2	湖北	本科	30000	离异	...
3	河南	本科	40000	单身	...

建表语句如下：

```
1  create table tb_user_desc (
2      id int primary key auto_increment,
3      city varchar(20),
4      edu varchar(10),
5      income int,
6      status char(2),
7      des varchar(100)
8  );
9
10 create table tb_user (
11     id int primary key auto_increment,
12     photo varchar(100),
13     nickname varchar(50),
```

```
    age int,
    gender char(1),
    desc_id int unique,
    -- 添加外键
    CONSTRAINT fk_user_desc FOREIGN KEY(desc_id) REFERENCES tb_user_desc(id)
);
```

查看表结构模型图：



2.5 数据库设计案例

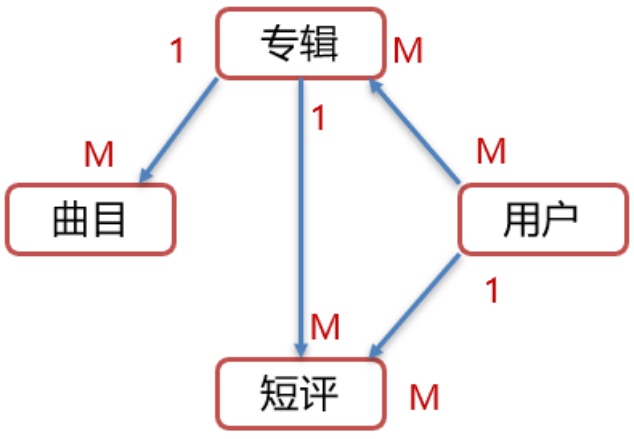
根据下图设计表及表和表之间的关系：

经过分析，我们分为 `专辑表` `曲目表` `短评表` `用户表` 4张表。

一个专辑可以有多个曲目，一个曲目只能属于某一张专辑，所以专辑表和曲目表的关系是**一对多**。

一个专辑可以被多个用户进行评论，一个用户可以对多个专辑进行评论，所以专辑表和用户表的关系是 **多对多**。

一个用户可以发多个短评，一个短评只能是某一个人发的，所以用户表和短评表的关系是 **一对多**。



3，多表查询

多表查询顾名思义就是从多张表中一次性的查询出我们想要的数据。我们通过具体的sql给他们演示，先准备环境

```sql
DROP TABLE IF EXISTS emp;
DROP TABLE IF EXISTS dept;


# 创建部门表
    CREATE TABLE dept(
        did INT PRIMARY KEY AUTO_INCREMENT,
        dname VARCHAR(20)
    );

    # 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
        dep_id INT,
        FOREIGN KEY (dep_id) REFERENCES dept(did) -- 外键，关联部门表(部门表的主键)
```

```sql
    );
    -- 添加部门数据
    INSERT INTO dept (dNAME) VALUES ('研发部'),('市场部'),('财务部'),('销售部');
    -- 添加员工数据
    INSERT INTO emp(NAME,gender,salary,join_date,dep_id) VALUES
    ('孙悟空','男',7200,'2013-02-24',1),
    ('猪八戒','男',3600,'2010-12-02',2),
    ('唐僧','男',9000,'2008-08-08',2),
    ('白骨精','女',5000,'2015-10-07',3),
    ('蜘蛛精','女',4500,'2011-03-14',1),
    ('小白龙','男',2500,'2011-02-14',null);
```

执行下面的多表查询语句

```sql
select * from emp , dept;  -- 从emp和dept表中查询所有的字段数据
```

结果如下：

信息 | 结果1 | 概况 | 状态

id	NAME	gender	salary	join_date	dep_id	did	dname
1	孙悟空	男	7200	2013-02-24	1	1	研发部
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	4	销售部
2	猪八戒	男	3600	2010-12-02	2	1	研发部
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

从上面的结果我们看到有一些无效的数据，如 `孙悟空` 这个员工属于1号部门，但也同时关联的2、3、4号部门。所以我们要通过限制员工表中的 `dep_id` 字段的值和部门表 `did` 字段的值相等来消除这些无效的数据，

```sql
select * from emp , dept where emp.dep_id = dept.did;
```

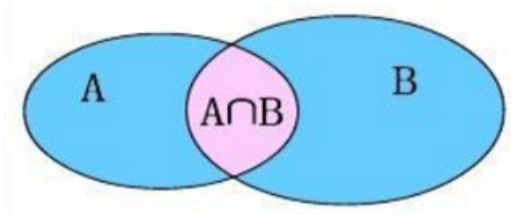
执行后结果如下：

信息 | 结果1 | 概况 | 状态

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

上面语句就是连接查询，那么多表查询都有哪些呢?

- 连接查询



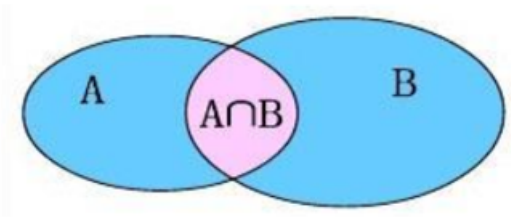
    - 内连接查询： 相当于查询AB交集数据
    - 外连接查询
        - 左外连接查询： 相当于查询A表所有数据和交集部门数据
        - 右外连接查询： 相当于查询B表所有数据和交集部分数据
- 子查询

3.1 内连接查询

- 语法

```sql
-- 隐式内连接
SELECT 字段列表 FROM 表1,表2… WHERE 条件;

-- 显示内连接
SELECT 字段列表 FROM 表1 [INNER] JOIN 表2 ON 条件;
```

> 内连接相当于查询 A B 交集数据



- 案例
  - 隐式内连接

```sql
SELECT
    *
FROM
    emp,
    dept
WHERE
    emp.dep_id = dept.did;
```

执行上述语句结果如下：

信息 | 结果1 | 概况 | 状态

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

  - 查询 emp的 name， gender，dept表的dname

```sql
SELECT
    emp. NAME,
    emp.gender,
    dept.dname
FROM
    emp,
    dept
WHERE
    emp.dep_id = dept.did;
```

执行语句结果如下：

信息 | 结果1 | 概况 | 状态

NAME	gender	dname
孙悟空	男	研发部
猪八戒	男	市场部
唐僧	男	市场部
白骨精	女	财务部
蜘蛛精	女	研发部

上面语句中使用表名指定字段所属有点麻烦，sql也支持给表指别名，上述语句可以改进为

```sql
SELECT
    t1. NAME,
    t1.gender,
    t2.dname
FROM
    emp t1,
    dept t2
WHERE
    t1.dep_id = t2.did;
```

- 显式内连接

```sql
select * from emp inner join dept on emp.dep_id = dept.did;
-- 上面语句中的inner可以省略，可以书写为如下语句
select * from emp  join dept on emp.dep_id = dept.did;
```

执行结果如下：

信息 | 结果1 | 概况 | 状态

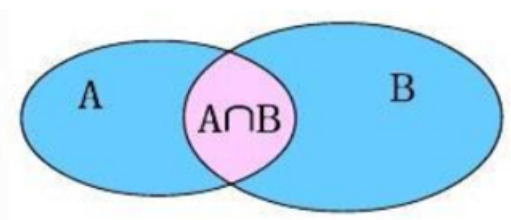
id	NAME	gender	salary	join_date	dep_id	did	dname
1	孙悟空	男	7200	2013-02-24	1	1	研发部
2	猪八戒	男	3600	2010-12-02	2	2	市场部
3	唐僧	男	9000	2008-08-08	2	2	市场部
4	白骨精	女	5000	2015-10-07	3	3	财务部
5	蜘蛛精	女	4500	2011-03-14	1	1	研发部

3.2 外连接查询

- 语法

```sql
-- 左外连接
SELECT 字段列表 FROM 表1 LEFT [OUTER] JOIN 表2 ON 条件;

-- 右外连接
SELECT 字段列表 FROM 表1 RIGHT [OUTER] JOIN 表2 ON 条件;
```

> 左外连接：相当于查询A表所有数据和交集部分数据
>
> 右外连接：相当于查询B表所有数据和交集部分数据



- 案例
  - 查询emp表所有数据和对应的部门信息（左外连接）

```sql
select * from emp left join dept on emp.dep_id = dept.did;
```

执行语句结果如下：

信息 | 结果1 | 概况 | 状态

id	NAME	gender	salary	join_date	dep_id	did	dname
1	孙悟空	男	7200	2013-02-24	1	1	研发部
5	蜘蛛精	女	4500	2011-03-14	1	1	研发部
2	猪八戒	男	3600	2010-12-02	2	2	市场部
3	唐僧	男	9000	2008-08-08	2	2	市场部
4	白骨精	女	5000	2015-10-07	3	3	财务部
6	小白龙	男	2500	2011-02-14	(Null)	(Null)	(Null)

结果显示查询到了左表（emp）中所有的数据及两张表能关联的数据。

  - 查询dept表所有数据和对应的员工信息（右外连接）

```sql
select * from emp right join dept on emp.dep_id = dept.did;
```

执行语句结果如下：

信息 | 结果1 | 概况 | 状态

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

结果显示查询到了右表（dept）中所有的数据及两张表能关联的数据。

要查询出部门表中所有的数据，也可以通过左外连接实现，只需要将两个表的位置进行互换：

```sql
select * from dept left join emp on emp.dep_id = dept.did;
```

3.3 子查询

- 概念

  查询中嵌套查询，称嵌套查询为子查询。

  什么是查询中嵌套查询呢？我们通过一个例子来看：

  需求：查询工资高于猪八戒的员工信息。

  来实现这个需求，我们就可以通过二步实现，第一步：先查询出来 猪八戒的工资

  ```sql
  select salary from emp where name = '猪八戒'
  ```

  第二步：查询工资高于猪八戒的员工信息

  ```sql
  select * from emp where salary > 3600;
  ```

  第二步中的3600可以通过第一步的sql查询出来，所以将3600用第一步的sql语句进行替换

  ```sql
  select * from emp where salary > (select salary from emp where name = '猪八戒');
  ```

  这就是查询语句中嵌套查询语句。

- 子查询根据查询结果不同，作用不同
  - 子查询语句结果是单行单列，子查询语句作为条件值，使用 = != > < 等进行条件判断
  - 子查询语句结果是多行单列，子查询语句作为条件值，使用 in 等关键字进行条件判断
  - 子查询语句结果是多行多列，子查询语句作为虚拟表
- 案例
  - 查询 '财务部' 和 '市场部' 所有的员工信息

    ```sql
    -- 查询 '财务部' 或者 '市场部' 所有的员工的部门did
    select did from dept where dname = '财务部' or dname = '市场部';

    select * from emp where dep_id in (select did from dept where dname = '财务部' or dname = '市场部');
    ```

  - 查询入职日期是 '2011-11-11' 之后的员工信息和部门信息

    ```sql
    -- 查询入职日期是 '2011-11-11' 之后的员工信息
    select * from emp where join_date > '2011-11-11' ;
    -- 将上面语句的结果作为虚拟表和dept表进行内连接查询
    select * from (select * from emp where join_date > '2011-11-11' ) t1, dept where t1.dep_id = dept.did;
    ```

3.4 案例

- 环境准备：

```sql
DROP TABLE IF EXISTS emp;
DROP TABLE IF EXISTS dept;
DROP TABLE IF EXISTS job;
DROP TABLE IF EXISTS salarygrade;

-- 部门表
CREATE TABLE dept (
  did INT PRIMARY KEY PRIMARY KEY, -- 部门id
  dname VARCHAR(50), -- 部门名称
  loc VARCHAR(50) -- 部门所在地
);

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

-- 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)
);
-- 工资等级表
CREATE TABLE salarygrade (
  grade INT PRIMARY KEY,   -- 级别
  losalary INT,  -- 最低工资
  hisalary INT -- 最高工资
);

-- 添加4个部门
INSERT INTO dept(did,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');

-- 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');


-- 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);


-- 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);
```

- 需求

1. 查询所有员工信息。查询员工编号，员工姓名，工资，职务名称，职务描述

```
/*
    分析:
        1. 员工编号，员工姓名，工资 信息在emp 员工表中
        2. 职务名称，职务描述 信息在 job 职务表中
        3. job 职务表 和 emp 员工表 是 一对多的关系 emp.job_id = job.id
*/
-- 方式一 ：隐式内连接
SELECT
    emp.id,
    emp.ename,
    emp.salary,
    job.jname,
    job.description
FROM
    emp,
    job
WHERE
    emp.job_id = job.id;

-- 方式二 ：显式内连接
SELECT
    emp.id,
    emp.ename,
    emp.salary,
    job.jname,
    job.description
FROM
    emp
INNER JOIN job ON emp.job_id = job.id;
```

2. 查询员工编号，员工姓名，工资，职务名称，职务描述，部门名称，部门位置

```
/*
    分析:
        1. 员工编号，员工姓名，工资 信息在emp 员工表中
        2. 职务名称，职务描述 信息在 job 职务表中
        3. job 职务表 和 emp 员工表 是 一对多的关系 emp.job_id = job.id

        4. 部门名称，部门位置 来自于 部门表 dept
        5. dept 和 emp 一对多关系 dept.id = emp.dept_id
*/

-- 方式一 ：隐式内连接
```

```sql
SELECT
    emp.id,
    emp.ename,
    emp.salary,
    job.jname,
    job.description,
    dept.dname,
    dept.loc
FROM
    emp,
    job,
    dept
WHERE
    emp.job_id = job.id
    and dept.id = emp.dept_id
;

-- 方式二 ： 显式内连接
SELECT
    emp.id,
    emp.ename,
    emp.salary,
    job.jname,
    job.description,
    dept.dname,
    dept.loc
FROM
    emp
INNER JOIN job ON emp.job_id = job.id
INNER JOIN dept ON dept.id = emp.dept_id
```

3. 查询员工姓名，工资，工资等级

```sql
/*
    分析:
        1. 员工姓名，工资 信息在emp 员工表中
        2. 工资等级 信息在 salarygrade 工资等级表中
        3. emp.salary >= salarygrade.losalary  and emp.salary <= salarygrade.hisalary
*/
SELECT
    emp.ename,
    emp.salary,
    t2.*
FROM
    emp,
    salarygrade t2
WHERE
    emp.salary >= t2.losalary
AND emp.salary <= t2.hisalary
```

4. 查询员工姓名，工资，职务名称，职务描述，部门名称，部门位置，工资等级

```sql
/*
    分析:
        1. 员工编号，员工姓名，工资 信息在emp 员工表中
        2. 职务名称，职务描述 信息在 job 职务表中
        3. job 职务表 和 emp 员工表 是 一对多的关系 emp.job_id = job.id

        4. 部门名称，部门位置 来自于 部门表 dept
        5. dept 和 emp 一对多关系 dept.id = emp.dept_id
        6. 工资等级 信息在 salarygrade 工资等级表中
        7. emp.salary >= salarygrade.losalary  and emp.salary <= salarygrade.hisalary
*/
SELECT
    emp.id,
```

```
    emp.ename,
    emp.salary,
    job.jname,
    job.description,
    dept.dname,
    dept.loc,
    t2.grade
FROM
    emp
INNER JOIN job ON emp.job_id = job.id
INNER JOIN dept ON dept.id = emp.dept_id
INNER JOIN salarygrade t2 ON emp.salary BETWEEN t2.losalary and t2.hisalary;
```

5. 查询出部门编号、部门名称、部门位置、部门人数

```
/*
    分析:
        1. 部门编号、部门名称、部门位置 来自于部门 dept 表
        2. 部门人数: 在emp表中 按照dept_id 进行分组，然后count(*)统计数量
        3. 使用子查询，让部门表和分组后的表进行内连接
*/
-- 根据部门id分组查询每一个部门id和员工数
select dept_id, count(*) from emp group by dept_id;

SELECT
    dept.id,
    dept.dname,
    dept.loc,
    t1.count
FROM
    dept,
    (
        SELECT
            dept_id,
            count(*) count
        FROM
            emp
        GROUP BY
            dept_id
    ) t1
WHERE
    dept.id = t1.dept_id
```

4，事务

4.1 概述

> 数据库的事务（Transaction）是一种机制、一个操作序列，包含了**一组数据库操作命令**。
>
> 事务把所有的命令作为一个整体一起向系统提交或撤销操作请求，即这一组数据库命令**要么同时成功，要么同时失败**。
>
> 事务是一个不可分割的工作逻辑单元。

这些概念不好理解，接下来举例说明，如下图有一张表

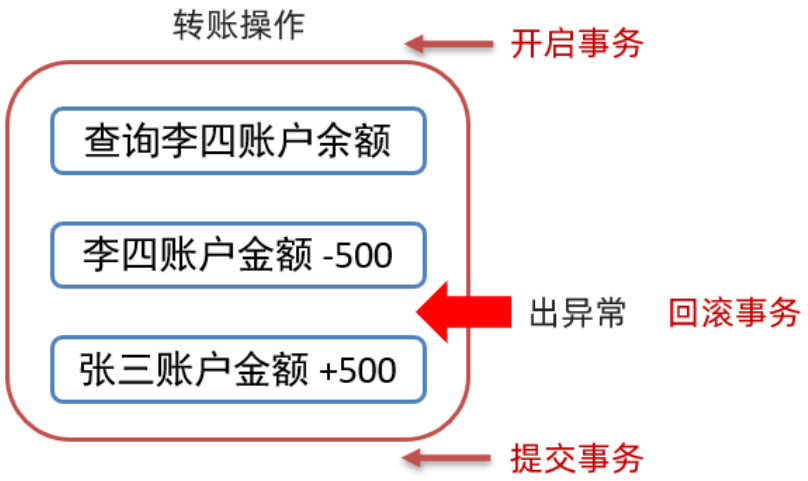
id	name	money
1	张三	1000
2	李四	500

张三和李四账户中各有100块钱，现李四需要转换500块钱给张三，具体的转账操作为

- 第一步：查询李四账户余额

- 第二步：从李四账户金额 -500
- 第三步：给张三账户金额 +500

现在假设在转账过程中第二步完成后出现了异常第三步没有执行，就会造成李四账户金额少了500，而张三金额并没有多500；这样的系统是有问题的。如果解决呢？使用事务可以解决上述问题



从上图可以看到在转账前开启事务，如果出现了异常回滚事务，三步正常执行就提交事务，这样就可以完美解决问题。

4.2 语法

- 开启事务

```
START TRANSACTION;
或者
BEGIN;
```

- 提交事务

```
commit;
```

- 回滚事务

```
rollback;
```

4.3 代码验证

- 环境准备

```
DROP TABLE IF EXISTS account;

-- 创建账户表
CREATE TABLE account(
    id int PRIMARY KEY auto_increment,
    name varchar(10),
    money double(10,2)
);

-- 添加数据
INSERT INTO account(name,money) values('张三',1000),('李四',1000);
```

- 不加事务演示问题

```
-- 转账操作
-- 1. 查询李四账户金额是否大于500

-- 2. 李四账户 -500
UPDATE account set money = money - 500 where name = '李四';

出现异常了...  -- 此处不是注释，在整体执行时会出问题，后面的sql则不执行
-- 3. 张三账户 +500
UPDATE account set money = money + 500 where name = '张三';
```

整体执行结果肯定会出问题，我们查询账户表中数据，发现李四账户少了500。

信息 | 结果1 | 概况 | 状态

id	name	money
1	张三	1000
2	李四	500

- 添加事务sql如下：

```sql
-- 开启事务
BEGIN;
-- 转账操作
-- 1. 查询李四账户金额是否大于500

-- 2. 李四账户 -500
UPDATE account set money = money - 500 where name = '李四';

出现异常了...  -- 此处不是注释，在整体执行时会出问题，后面的sql则不执行
-- 3. 张三账户 +500
UPDATE account set money = money + 500 where name = '张三';

-- 提交事务
COMMIT;

-- 回滚事务
ROLLBACK;
```

上面sql中的执行成功进选择执行提交事务，而出现问题则执行回滚事务的语句。以后我们肯定不可能这样操作，而是在java中进行操作，在java中可以抓取异常，没出现异常提交事务，出现异常回滚事务。

4.4 事务的四大特征

- 原子性（Atomicity）：事务是不可分割的最小操作单位，要么同时成功，要么同时失败
- 一致性（Consistency）：事务完成时，必须使所有的数据都保持一致状态
- 隔离性（Isolation）：多个事务之间，操作的可见性
- 持久性（Durability）：事务一旦提交或回滚，它对数据库中的数据的改变就是永久的

说明：

mysql中事务是自动提交的。

也就是说我们不添加事务执行sql语句，语句执行完毕会自动的提交事务。

可以通过下面语句查询默认提交方式：

```sql
SELECT @@autocommit;
```

查询到的结果是1 则表示自动提交，结果是0表示手动提交。当然也可以通过下面语句修改提交方式

```sql
set @@autocommit = 0;
```