MySQL进阶-02-授课笔记

一、约束

1.外键约束

- 外键约束概念
  - 让表和表之间产生关系，从而保证数据的准确性！
- 建表时添加外键约束
  - 为什么要有外键约束

```
-- 创建db2数据库
CREATE DATABASE db2;
-- 使用db2数据库
USE db2;

-- 创建user用户表
CREATE TABLE USER(
    id INT PRIMARY KEY AUTO_INCREMENT,     -- id
    NAME VARCHAR(20) NOT NULL              -- 姓名
);
-- 添加用户数据
INSERT INTO USER VALUES (NULL,'张三'),(NULL,'李四'),(NULL,'王五');

-- 创建orderlist订单表
CREATE TABLE orderlist(
    id INT PRIMARY KEY AUTO_INCREMENT,     -- id
    number VARCHAR(20) NOT NULL,           -- 订单编号
    uid INT                                -- 订单所属用户
);
-- 添加订单数据
INSERT INTO orderlist VALUES (NULL,'hm001',1),(NULL,'hm002',1),
(NULL,'hm003',2),(NULL,'hm004',2),
(NULL,'hm005',3),(NULL,'hm006',3);

-- 添加一个订单，但是没有所属用户。这合理吗？
INSERT INTO orderlist VALUES (NULL,'hm007',8);
-- 删除王五这个用户，但是订单表中王五还有很多个订单呢。这合理吗？
DELETE FROM USER WHERE NAME='王五';

-- 所以我们需要添加外键约束，让两张表产生关系
```

  - 外键约束格式

```
CONSTRAINT 外键名 FOREIGN KEY (本表外键列名) REFERENCES 主表名(主表主键列名)
```

  - 创建表添加外键约束

```
-- 创建user用户表
CREATE TABLE USER(
    id INT PRIMARY KEY AUTO_INCREMENT,     -- id
```

```
    NAME VARCHAR(20) NOT NULL              -- 姓名
);
-- 添加用户数据
INSERT INTO USER VALUES (NULL,'张三'),(NULL,'李四'),(NULL,'王五');

-- 创建orderlist订单表
CREATE TABLE orderlist(
    id INT PRIMARY KEY AUTO_INCREMENT,     -- id
    number VARCHAR(20) NOT NULL,           -- 订单编号
    uid INT,                               -- 订单所属用户
    CONSTRAINT ou_fk1 FOREIGN KEY (uid) REFERENCES USER(id)   -- 添加外键约束
);
-- 添加订单数据
INSERT INTO orderlist VALUES (NULL,'hm001',1),(NULL,'hm002',1),
(NULL,'hm003',2),(NULL,'hm004',2),
(NULL,'hm005',3),(NULL,'hm006',3);

-- 添加一个订单，但是没有所属用户。无法添加
INSERT INTO orderlist VALUES (NULL,'hm007',8);
-- 删除王五这个用户，但是订单表中王五还有很多个订单呢。无法删除
DELETE FROM USER WHERE NAME='王五';
```

- 删除外键约束

```
-- 标准语法
ALTER TABLE 表名 DROP FOREIGN KEY 外键名;

-- 删除外键
ALTER TABLE orderlist DROP FOREIGN KEY ou_fk1;
```

- 建表后添加外键约束

```
-- 标准语法
ALTER TABLE 表名 ADD CONSTRAINT 外键名 FOREIGN KEY (本表外键列名) REFERENCES 主表名
(主键列名);

-- 添加外键约束
ALTER TABLE orderlist ADD CONSTRAINT ou_fk1 FOREIGN KEY (uid) REFERENCES
USER(id);
```

2.外键的级联更新和级联删除(了解)

- 什么是级联更新和级联删除
  - 当我想把user用户表中的某个用户删掉，我希望该用户所有的订单也随之被删除
  - 当我想把user用户表中的某个用户id修改，我希望订单表中该用户所属的订单用户编号也随之修改
- 添加级联更新和级联删除

```
-- 添加外键约束，同时添加级联更新   标准语法
ALTER TABLE 表名 ADD CONSTRAINT 外键名 FOREIGN KEY (本表外键列名) REFERENCES 主表名
(主键列名) ON UPDATE CASCADE;

-- 添加外键约束，同时添加级联删除   标准语法
ALTER TABLE 表名 ADD CONSTRAINT 外键名 FOREIGN KEY (本表外键列名) REFERENCES 主表名
(主键列名) ON DELETE CASCADE;
```

```
-- 添加外键约束，同时添加级联更新和级联删除  标准语法
ALTER TABLE 表名 ADD CONSTRAINT 外键名 FOREIGN KEY (本表外键列名) REFERENCES 主表名(主键列名) ON UPDATE CASCADE ON DELETE CASCADE;


-- 删除外键约束
ALTER TABLE orderlist DROP FOREIGN KEY ou_fk1;

-- 添加外键约束，同时添加级联更新和级联删除
ALTER TABLE orderlist ADD CONSTRAINT ou_fk1 FOREIGN KEY (uid) REFERENCES USER(id) ON UPDATE CASCADE ON DELETE CASCADE;

-- 将王五用户的id修改为5     订单表中的uid也随之被修改
UPDATE USER SET id=5 WHERE id=3;

-- 将王五用户删除       订单表中该用户所有订单也随之删除
DELETE FROM USER WHERE id=5;
```

二、多表设计

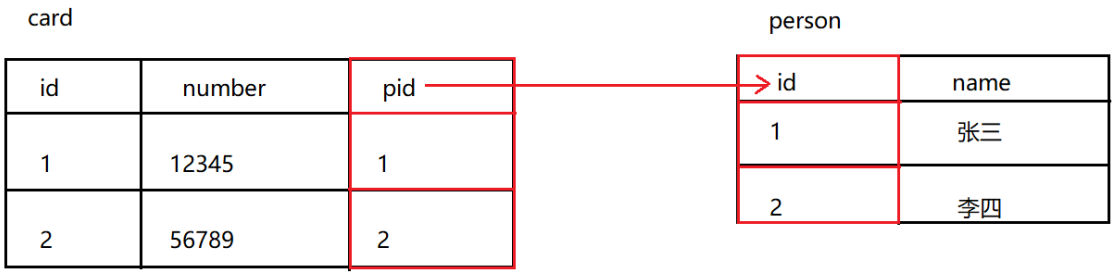
1.一对一(了解)

- 分析
  - 人和身份证。一个人只有一个身份证，一个身份证只能对应一个人!
- 实现原则
  - 在任意一个表建立外键，去关联另外一个表的主键
- SQL演示

```
-- 创建db5数据库
CREATE DATABASE db5;
-- 使用db5数据库
USE db5;

-- 创建person表
CREATE TABLE person(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(20)
);
-- 添加数据
INSERT INTO person VALUES (NULL,'张三'),(NULL,'李四');

-- 创建card表
CREATE TABLE card(
    id INT PRIMARY KEY AUTO_INCREMENT,
    number VARCHAR(50),
    pid INT UNIQUE,
    CONSTRAINT cp_fk1 FOREIGN KEY (pid) REFERENCES person(id) -- 添加外键
);
-- 添加数据
INSERT INTO card VALUES (NULL,'12345',1),(NULL,'56789',2);
```

- 图解

一对一



实现原则：

在任意一个表建立外键，去关联另外一个表的主键

2.一对多

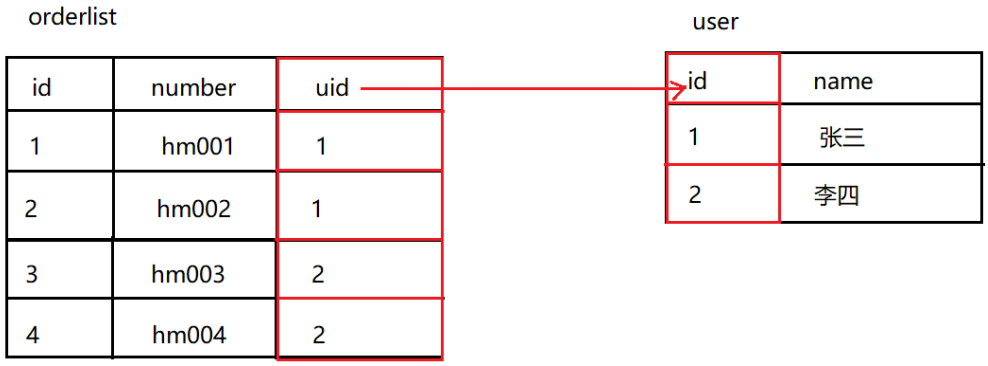
- 分析
  - 用户和订单。一个用户可以有多个订单！
  - 商品分类和商品。一个分类下可以有多个商品！
- 实现原则
  - 在多的一方，建立外键约束，来关联一的一方主键
- SQL演示

```
/*
    用户和订单
*/
-- 创建user表
CREATE TABLE USER(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(20)
);
-- 添加数据
INSERT INTO USER VALUES (NULL,'张三'),(NULL,'李四');

-- 创建orderlist表
CREATE TABLE orderlist(
    id INT PRIMARY KEY AUTO_INCREMENT,
    number VARCHAR(20),
    uid INT,
    CONSTRAINT ou_fk1 FOREIGN KEY (uid) REFERENCES USER(id)  -- 添加外键约束
);
-- 添加数据
INSERT INTO orderlist VALUES (NULL,'hm001',1),(NULL,'hm002',1),(NULL,'hm003',2),
(NULL,'hm004',2);


/*
    商品分类和商品
*/
-- 创建category表
CREATE TABLE category(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(10)
);
-- 添加数据
INSERT INTO category VALUES (NULL,'手机数码'),(NULL,'电脑办公');
```

```sql
-- 创建product表
CREATE TABLE product(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(30),
    cid INT,
    CONSTRAINT pc_fk1 FOREIGN KEY (cid) REFERENCES category(id)  -- 添加外键约束
);
-- 添加数据
INSERT INTO product VALUES (NULL,'华为P30',1),(NULL,'小米note3',1),
(NULL,'联想电脑',2),(NULL,'苹果电脑',2);
```

- 图解

一对多



实现原则：

在多的一方，建立外键约束，来关联一的一方主键

3.多对多

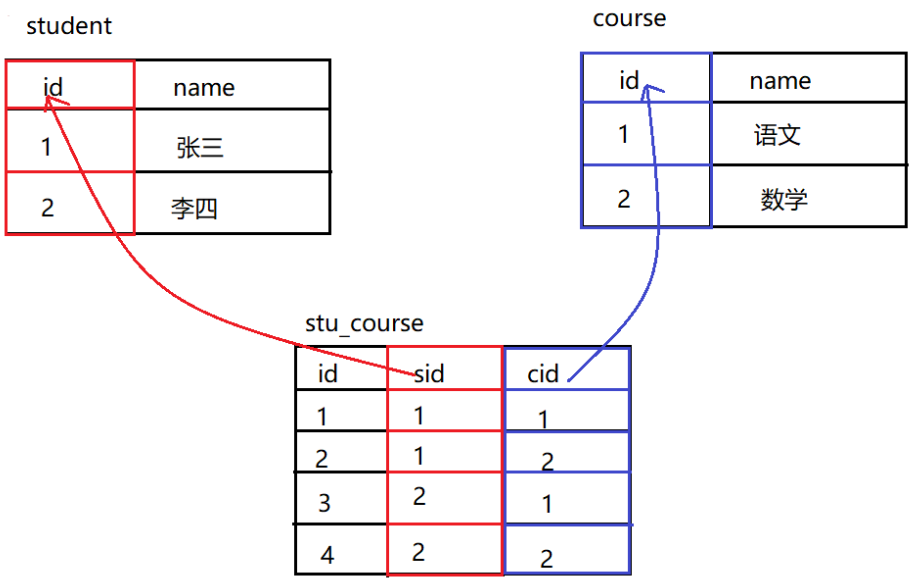
- 分析
  - 学生和课程。一个学生可以选择多个课程，一个课程也可以被多个学生选择!
- 实现原则
  - 需要借助第三张表中间表，中间表至少包含两个列，这两个列作为中间表的外键，分别关联两张表的主键
- SQL演示

```sql
-- 创建student表
CREATE TABLE student(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(20)
);
-- 添加数据
INSERT INTO student VALUES (NULL,'张三'),(NULL,'李四');

-- 创建course表
CREATE TABLE course(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(10)
);
-- 添加数据
INSERT INTO course VALUES (NULL,'语文'),(NULL,'数学');

-- 创建中间表
CREATE TABLE stu_course(
    id INT PRIMARY KEY AUTO_INCREMENT,
```

```
    sid INT, -- 用于和student表的id进行外键关联
    cid INT, -- 用于和course表的id进行外键关联
    CONSTRAINT sc_fk1 FOREIGN KEY (sid) REFERENCES student(id), -- 添加外键约束
    CONSTRAINT sc_fk2 FOREIGN KEY (cid) REFERENCES course(id)   -- 添加外键约束
);
-- 添加数据
INSERT INTO stu_course VALUES (NULL,1,1),(NULL,1,2),(NULL,2,1),(NULL,2,2);
```

- 图解

多对多



实现原则：

需要借助第三张表中间表，中间表至少包含两个列，这两个列作为中间表的外键，分别关联两张表的主键

三、多表查询

1.多表查询-数据准备

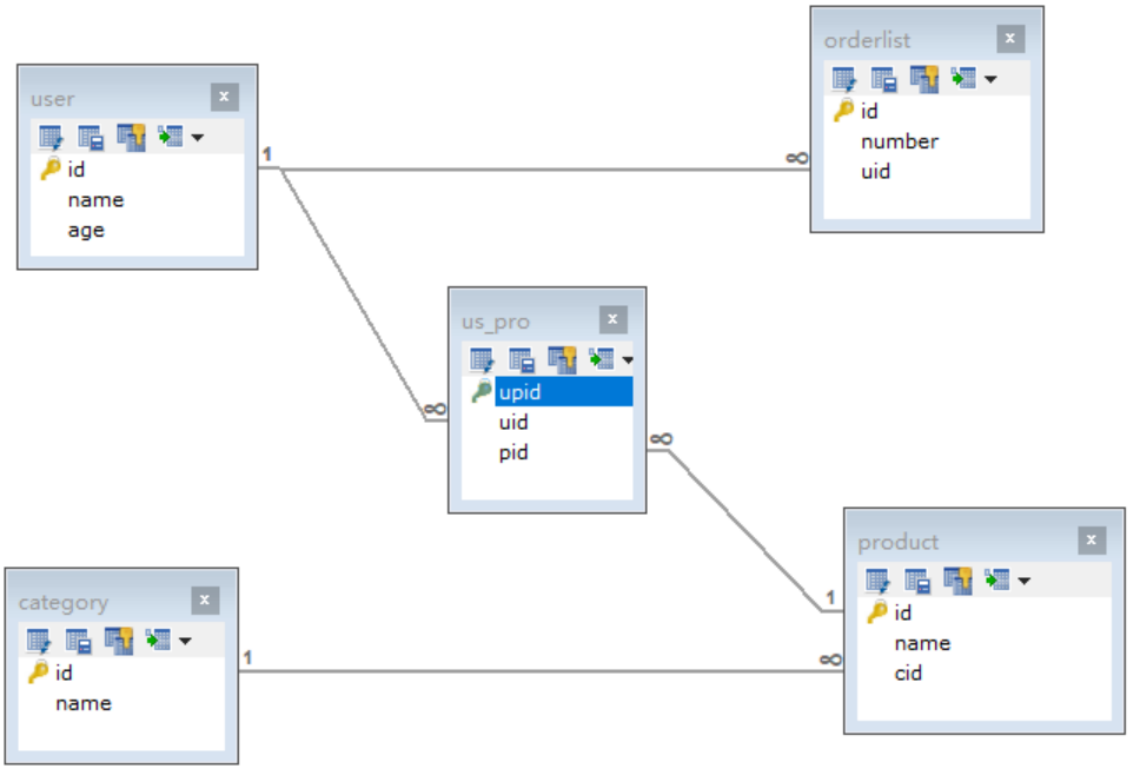
- SQL语句

```
-- 创建db6数据库
CREATE DATABASE db6;
-- 使用db6数据库
USE db6;

-- 创建user表
CREATE TABLE USER(
    id INT PRIMARY KEY AUTO_INCREMENT,  -- 用户id
    NAME VARCHAR(20),                   -- 用户姓名
    age INT                             -- 用户年龄
);
-- 添加数据
INSERT INTO USER VALUES (1,'张三',23);
INSERT INTO USER VALUES (2,'李四',24);
INSERT INTO USER VALUES (3,'王五',25);
INSERT INTO USER VALUES (4,'赵六',26);


-- 订单表
CREATE TABLE orderlist(
    id INT PRIMARY KEY AUTO_INCREMENT,  -- 订单id
    number VARCHAR(30),                 -- 订单编号
```

```
    uid INT,     -- 外键字段
    CONSTRAINT ou_fk1 FOREIGN KEY (uid) REFERENCES USER(id)
);
-- 添加数据
INSERT INTO orderlist VALUES (1,'hm001',1);
INSERT INTO orderlist VALUES (2,'hm002',1);
INSERT INTO orderlist VALUES (3,'hm003',2);
INSERT INTO orderlist VALUES (4,'hm004',2);
INSERT INTO orderlist VALUES (5,'hm005',3);
INSERT INTO orderlist VALUES (6,'hm006',3);
INSERT INTO orderlist VALUES (7,'hm007',NULL);


-- 商品分类表
CREATE TABLE category(
    id INT PRIMARY KEY AUTO_INCREMENT,  -- 商品分类id
    NAME VARCHAR(10)                    -- 商品分类名称
);
-- 添加数据
INSERT INTO category VALUES (1,'手机数码');
INSERT INTO category VALUES (2,'电脑办公');
INSERT INTO category VALUES (3,'烟酒茶糖');
INSERT INTO category VALUES (4,'鞋靴箱包');


-- 商品表
CREATE TABLE product(
    id INT PRIMARY KEY AUTO_INCREMENT,   -- 商品id
    NAME VARCHAR(30),                    -- 商品名称
    cid INT, -- 外键字段
    CONSTRAINT cp_fk1 FOREIGN KEY (cid) REFERENCES category(id)
);
-- 添加数据
INSERT INTO product VALUES (1,'华为手机',1);
INSERT INTO product VALUES (2,'小米手机',1);
INSERT INTO product VALUES (3,'联想电脑',2);
INSERT INTO product VALUES (4,'苹果电脑',2);
INSERT INTO product VALUES (5,'中华香烟',3);
INSERT INTO product VALUES (6,'玉溪香烟',3);
INSERT INTO product VALUES (7,'计生用品',NULL);


-- 中间表
CREATE TABLE us_pro(
    upid INT PRIMARY KEY AUTO_INCREMENT,  -- 中间表id
    uid INT, -- 外键字段。需要和用户表的主键产生关联
    pid INT, -- 外键字段。需要和商品表的主键产生关联
    CONSTRAINT up_fk1 FOREIGN KEY (uid) REFERENCES USER(id),
    CONSTRAINT up_fk2 FOREIGN KEY (pid) REFERENCES product(id)
);
-- 添加数据
INSERT INTO us_pro VALUES (NULL,1,1);
INSERT INTO us_pro VALUES (NULL,1,2);
INSERT INTO us_pro VALUES (NULL,1,3);
INSERT INTO us_pro VALUES (NULL,1,4);
INSERT INTO us_pro VALUES (NULL,1,5);
INSERT INTO us_pro VALUES (NULL,1,6);
INSERT INTO us_pro VALUES (NULL,1,7);
```

```
INSERT INTO us_pro VALUES (NULL,2,1);
INSERT INTO us_pro VALUES (NULL,2,2);
INSERT INTO us_pro VALUES (NULL,2,3);
INSERT INTO us_pro VALUES (NULL,2,4);
INSERT INTO us_pro VALUES (NULL,2,5);
INSERT INTO us_pro VALUES (NULL,2,6);
INSERT INTO us_pro VALUES (NULL,2,7);
INSERT INTO us_pro VALUES (NULL,3,1);
INSERT INTO us_pro VALUES (NULL,3,2);
INSERT INTO us_pro VALUES (NULL,3,3);
INSERT INTO us_pro VALUES (NULL,3,4);
INSERT INTO us_pro VALUES (NULL,3,5);
INSERT INTO us_pro VALUES (NULL,3,6);
INSERT INTO us_pro VALUES (NULL,3,7);
INSERT INTO us_pro VALUES (NULL,4,1);
INSERT INTO us_pro VALUES (NULL,4,2);
INSERT INTO us_pro VALUES (NULL,4,3);
INSERT INTO us_pro VALUES (NULL,4,4);
INSERT INTO us_pro VALUES (NULL,4,5);
INSERT INTO us_pro VALUES (NULL,4,6);
INSERT INTO us_pro VALUES (NULL,4,7);
```

- 架构器图解



2.多表查询-笛卡尔积查询(了解)

- 有两张表，获取这两个表的所有组合情况
- 要完成多表查询，需要消除这些没有用的数据
- 多表查询格式

```
SELECT
    列名列表
FROM
    表名列表
WHERE
    条件...
```

- 笛卡尔积查询

```
-- 标准语法
SELECT 列名 FROM 表名1,表名2,...;

-- 查询user表和orderlist表
SELECT * FROM USER,orderlist;
```

3.多表查询-内连接查询

- 查询原理
    - 内连接查询的是两张表有交集的部分数据(有主外键关联的数据)
- 显式内连接

```
-- 标准语法
SELECT 列名 FROM 表名1 [INNER] JOIN 表名2 ON 条件;

-- 查询用户信息和对应的订单信息
SELECT * FROM USER INNER JOIN orderlist ON user.id=orderlist.uid;
SELECT * FROM USER JOIN orderlist ON user.id=orderlist.uid;

-- 查询用户信息和对应的订单信息，起别名
SELECT * FROM USER u JOIN orderlist o ON u.id=o.uid;

-- 查询用户姓名，年龄。和订单编号
SELECT
    u.`name`,    -- 姓名
    u.`age`,     -- 年龄
    o.`number`   -- 订单编号
FROM
    USER u           -- 用户表
JOIN
    orderlist o      -- 订单表
ON
    u.`id` = o.`uid`;
```

- 隐式内连接

```sql
-- 标准语法
SELECT 列名 FROM 表名1,表名2 WHERE 条件;

-- 查询用户姓名，年龄。和订单编号
SELECT
    u.`name`,    -- 姓名
    u.`age`,     -- 年龄
    o.`number`   -- 订单编号
FROM
    USER u,      -- 用户表
    orderlist o      -- 订单表
WHERE
    u.`id`=o.`uid`;
```

4.多表查询-外连接查询

- 左外连接
  - 查询原理
    - 查询左表的全部数据，和左右两张表有交集部分的数据
  - 基本演示

```sql
-- 标准语法
SELECT 列名 FROM 表名1 LEFT [OUTER] JOIN 表名2 ON 条件;

-- 查询所有用户信息，以及用户对应的订单信息
SELECT
    u.`name`,    -- 姓名
    u.`age`,     -- 年龄
    o.`number`   -- 订单编号
FROM
    USER u           -- 用户表
LEFT OUTER JOIN
    orderlist o      -- 订单表
ON
    u.`id`=o.`uid`;
```

- 右外连接
  - 查询原理
    - 查询右表的全部数据，和左右两张表有交集部分的数据
  - 基本演示

```sql
-- 基本语法
SELECT 列名 FROM 表名1 RIGHT [OUTER] JOIN 表名2 ON 条件;

-- 查询所有订单信息，以及订单所属的用户信息
SELECT
    u.`name`,    -- 姓名
    u.`age`,     -- 年龄
    o.`number`   -- 订单编号
FROM
    USER u           -- 用户表
RIGHT OUTER JOIN
    orderlist o      -- 订单表
ON
```

```
    u.`id`=o.`uid`;
```

5.多表查询-子查询

- 子查询介绍
  - 查询语句中嵌套了查询语句。我们就将嵌套查询称为子查询!
- 子查询-结果是单行单列的
  - 可以作为条件，使用运算符进行判断!
  - 基本演示

```
-- 标准语法
SELECT 列名 FROM 表名 WHERE 列名=(SELECT 聚合函数(列名) FROM 表名 [WHERE 条件]);

-- 查询年龄最高的用户姓名
SELECT MAX(age) FROM USER;                -- 查询出最高年龄
SELECT NAME,age FROM USER WHERE age=26; -- 根据查询出来的最高年龄，查询姓名和年龄
SELECT NAME,age FROM USER WHERE age = (SELECT MAX(age) FROM USER);
```

- 子查询-结果是多行单列的
  - 可以作为条件，使用运算符in或not in进行判断!
  - 基本演示

```
-- 标准语法
SELECT 列名 FROM 表名 WHERE 列名 [NOT] IN (SELECT 列名 FROM 表名 [WHERE 条件]);

-- 查询张三和李四的订单信息
SELECT id FROM USER WHERE NAME='张三' OR NAME='李四';   -- 查询张三和李四用户的
id
SELECT number,uid FROM orderlist WHERE uid=1 OR uid=2; -- 根据id查询订单
SELECT number,uid FROM orderlist WHERE uid IN (SELECT id FROM USER WHERE
NAME='张三' OR NAME='李四');
```

- 子查询-结果是多行多列的
  - 可以作为一张虚拟表参与查询!
  - 基本演示

```
-- 标准语法
SELECT 列名 FROM 表名 [别名],(SELECT 列名 FROM 表名 [WHERE 条件]) [别名] [WHERE
条件];

-- 查询订单表中id大于4的订单信息和所属用户信息
SELECT * FROM USER u,(SELECT * FROM orderlist WHERE id>4) o WHERE
u.id=o.uid;
```

6.多表查询练习

- 查询用户的编号、姓名、年龄。订单编号

```
/*
分析:
    用户的编号、姓名、年龄  user表      订单编号 orderlist表
    条件：user.id = orderlist.uid
*/
```

```
SELECT
    t1.`id`,    -- 用户编号
    t1.`name`,  -- 用户姓名
    t1.`age`,   -- 用户年龄
    t2.`number` -- 订单编号
FROM
    USER t1,        -- 用户表
    orderlist t2    -- 订单表
WHERE
    t1.`id` = t2.`uid`;
```

- 查询所有的用户。用户的编号、姓名、年龄。订单编号

```
/*
分析：
    用户的编号、姓名、年龄 user表      订单编号 orderlist表
    条件：user.id = orderlist.uid
    查询所有用户，使用左外连接
*/
SELECT
    t1.`id`,    -- 用户编号
    t1.`name`,  -- 用户姓名
    t1.`age`,   -- 用户年龄
    t2.`number` -- 订单编号
FROM
    USER t1         -- 用户表
LEFT OUTER JOIN
    orderlist t2    -- 订单表
ON
    t1.`id` = t2.`uid`;
```

- 查询所有的订单。用户的编号、姓名、年龄。订单编号

```
/*
分析：
    用户的编号、姓名、年龄 user表      订单编号 orderlist表
    条件：user.id = orderlist.uid
    查询所有订单，使用右外连接
*/
SELECT
    t1.`id`,    -- 用户编号
    t1.`name`,  -- 用户姓名
    t1.`age`,   -- 用户年龄
    t2.`number` -- 订单编号
FROM
    USER t1          -- 用户表
RIGHT OUTER JOIN
    orderlist t2     -- 订单表
ON
    t1.`id` = t2.`uid`;
```

- 查询用户年龄大于23岁的信息。显示用户的编号、姓名、年龄。订单编号

```
/*
分析：
    用户的编号、姓名、年龄 user表      订单编号 orderlist表
```

```
    条件：user.age > 23 AND user.id = orderlist.uid
*/
/*
select
    t1.`id`,    -- 用户编号
    t1.`name`,  -- 用户姓名
    t1.`age`,   -- 用户年龄
    t2.`number` -- 订单编号
from
    user t1,     -- 用户表
    orderlist t2 -- 订单表
where
    t1.`age` > 23
    and
    t1.`id` = t2.`uid`;
*/
SELECT
    t1.`id`,    -- 用户编号
    t1.`name`,  -- 用户姓名
    t1.`age`,   -- 用户年龄
    t2.`number` -- 订单编号
FROM
    USER t1       -- 用户表
LEFT OUTER JOIN
    orderlist t2  -- 订单表
ON
    t1.`id` = t2.`uid`
WHERE
    t1.`age` > 23;
```

- 查询张三和李四用户的信息。显示用户的编号、姓名、年龄。订单编号

```
/*
分析：
    用户的编号、姓名、年龄 user表      订单编号 orderlist表
    条件：user.id = orderlist.uid AND user.name IN ('张三','李四');
*/
SELECT
    t1.`id`,    -- 用户编号
    t1.`name`,  -- 用户姓名
    t1.`age`,   -- 用户年龄
    t2.`number` -- 订单编号
FROM
    USER t1,        -- 用户表
    orderlist t2    -- 订单表
WHERE
    t1.`id` = t2.`uid`
    AND
    -- (t1.`name` = '张三' OR t1.`name` = '李四');
    t1.`name` IN ('张三','李四');
```

- 查询商品分类的编号、分类名称。分类下的商品名称

```
/*
分析：
    商品分类的编号、分类名称 category表      分类下的商品名称 product表
```

```
    条件：category.id = product.cid
*/
SELECT
    t1.`id`,    -- 分类编号
    t1.`name`,  -- 分类名称
    t2.`name`   -- 商品名称
FROM
    category t1,    -- 商品分类表
    product t2      -- 商品表
WHERE
    t1.`id` = t2.`cid`;
```

- 查询所有的商品分类。商品分类的编号、分类名称。分类下的商品名称

```
/*
分析：
    商品分类的编号、分类名称 category表      分类下的商品名称 product表
    条件：category.id = product.cid
    查询所有的商品分类，使用左外连接
*/
SELECT
    t1.`id`,    -- 分类编号
    t1.`name`,  -- 分类名称
    t2.`name`   -- 商品名称
FROM
    category t1 -- 商品分类表
LEFT OUTER JOIN
    product t2  -- 商品表
ON
    t1.`id` = t2.`cid`;
```

- 查询所有的商品信息。商品分类的编号、分类名称。分类下的商品名称

```
/*
分析：
    商品分类的编号、分类名称 category表      分类下的商品名称 product表
    条件：category.id = product.cid
    查询所有的商品信息，使用右外连接
*/
SELECT
    t1.`id`,    -- 分类编号
    t1.`name`,  -- 分类名称
    t2.`name`   -- 商品名称
FROM
    category t1 -- 商品分类表
RIGHT OUTER JOIN
    product t2  -- 商品表
ON
    t1.`id` = t2.`cid`;
```

- 查询所有的用户和所有的商品。显示用户的编号、姓名、年龄。商品名称

```
/*
分析：
    用户的编号、姓名、年龄 user表    商品名称 product表    中间表 us_pro
    条件：us_pro.uid = user.id AND us_pro.pid = product.id
```

```
*/
SELECT
    t1.`id`,    -- 用户编号
    t1.`name`,  -- 用户名称
    t1.`age`,   -- 用户年龄
    t2.`name`   -- 商品名称
FROM
    USER t1,    -- 用户表
    product t2, -- 商品表
    us_pro t3   -- 中间表
WHERE
    t3.`uid` = t1.`id`
    AND
    t3.`pid` = t2.`id`;
```

- 查询张三和李四这两个用户可以看到的商品。显示用户的编号、姓名、年龄。商品名称

```
/*
分析：
    用户的编号、姓名、年龄 user表    商品名称 product表    中间表 us_pro
    条件：us_pro.uid = user.id AND us_pro.pid = product.id AND user.name IN ('张三','李四')
*/
SELECT
    t1.`id`,    -- 用户编号
    t1.`name`,  -- 用户名称
    t1.`age`,   -- 用户年龄
    t2.`name`   -- 商品名称
FROM
    USER t1,    -- 用户表
    product t2, -- 商品表
    us_pro t3   -- 中间表
WHERE
    (t3.`uid` = t1.`id` AND t3.`pid` = t2.`id`)
    AND
    -- (t1.`name` = '张三' or t1.`name` = '李四');
    t1.`name` IN ('张三','李四');
```

7.多表查询-自关联查询

- 自关联查询介绍
    - 同一张表中有数据关联。可以多次查询这同一个表!
- 自关联查询演示

```
-- 创建员工表
CREATE TABLE employee(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(20),
    mgr INT,
    salary DOUBLE
);
-- 添加数据
INSERT INTO employee VALUES (1001,'孙悟空',1005,9000.00),
(1002,'猪八戒',1005,8000.00),
(1003,'沙和尚',1005,8500.00),
(1004,'小白龙',1005,7900.00),
```

```
(1005,'唐僧',NULL,15000.00),
(1006,'武松',1009,7600.00),
(1007,'李逵',1009,7400.00),
(1008,'林冲',1009,8100.00),
(1009,'宋江',NULL,16000.00);

-- 查询所有员工的姓名及其直接上级的姓名，没有上级的员工也需要查询
/*
分析：
    员工姓名 employee表         直接上级姓名 employee表
    条件：employee.mgr = employee.id
    查询左表的全部数据，和左右两张表交集部分数据，使用左外连接
*/
SELECT
    t1.name,    -- 员工姓名
    t1.mgr,     -- 上级编号
    t2.id,      -- 员工编号
    t2.name     -- 员工姓名
FROM
    employee t1  -- 员工表
LEFT OUTER JOIN
    employee t2  -- 员工表
ON
    t1.mgr = t2.id;
```

四、视图

1.视图的概念

- 视图是一种虚拟存在的数据表
- 这个虚拟的表并不在数据库中实际存在
- 作用是将一些比较复杂的查询语句的结果，封装到一个虚拟表中。后期再有相同复杂查询时，直接查询这张虚拟表即可
- 说白了，视图就是将一条SELECT查询语句的结果封装到了一个虚拟表中，所以我们在创建视图的时候，工作重心就要放在这条SELECT查询语句上

2.视图的好处

- 简单
    - 对于使用视图的用户不需要关心表的结构、关联条件和筛选条件。因为这张虚拟表中保存的就是已经过滤好条件的结果集
- 安全
    - 视图可以设置权限，致使访问视图的用户只能访问他们被允许查询的结果集
- 数据独立
    - 一旦视图的结构确定了，可以屏蔽表结构变化对用户的影响，源表增加列对视图没有影响；源表修改列名，则可以通过修改视图来解决，不会造成对访问者的影响

3.视图数据准备

```
-- 创建db7数据库
CREATE DATABASE db7;

-- 使用db7数据库
USE db7;
```

```
-- 创建country表
CREATE TABLE country(
    id INT PRIMARY KEY AUTO_INCREMENT,
    country_name VARCHAR(30)
);
-- 添加数据
INSERT INTO country VALUES (NULL,'中国'),(NULL,'美国'),(NULL,'俄罗斯');

-- 创建city表
CREATE TABLE city(
    id INT PRIMARY KEY AUTO_INCREMENT,
    city_name VARCHAR(30),
    cid INT, -- 外键列。关联country表的主键列id
    CONSTRAINT cc_fk1 FOREIGN KEY (cid) REFERENCES country(id)
);
-- 添加数据
INSERT INTO city VALUES (NULL,'北京',1),(NULL,'上海',1),(NULL,'纽约',2),(NULL,'莫斯科',3);
```

4.视图的创建

- 创建视图语法

```
-- 标准语法
CREATE VIEW 视图名称 [(列名列表)] AS 查询语句;
```

- 普通多表查询，查询城市和所属国家

```
-- 普通多表查询，查询城市和所属国家
SELECT
    t1.*,
    t2.country_name
FROM
    city t1,
    country t2
WHERE
    t1.cid = t2.id;

-- 经常需要查询这样的数据，就可以创建一个视图
```

- 创建视图基本演示

```
-- 创建一个视图。将查询出来的结果保存到这张虚拟表中
CREATE
VIEW
    city_country
AS
    SELECT t1.*,t2.country_name FROM city t1,country t2 WHERE t1.cid=t2.id;
```

- 创建视图并指定列名基本演示

```
-- 创建一个视图，指定列名。将查询出来的结果保存到这张虚拟表中
CREATE
VIEW
    city_country2 (city_id,city_name,cid,country_name)
AS
    SELECT t1.*,t2.country_name FROM city t1,country t2 WHERE t1.cid=t2.id;
```

5.视图的查询

- 查询视图语法

```
-- 标准语法
SELECT * FROM 视图名称;
```

- 查询视图基本演示

```
-- 查询视图。查询这张虚拟表，就等效于查询城市和所属国家
SELECT * FROM city_country;

-- 查询指定列名的视图
SELECT * FROM city_country2;

-- 查询所有数据表，视图也会查询出来
SHOW TABLES;
```

- 查询视图创建语法

```
-- 标准语法
SHOW CREATE VIEW 视图名称;
```

- 查询视图创建语句基本演示

```
SHOW CREATE VIEW city_country;
```

6.视图的修改

- 修改视图表中的数据

```
-- 标准语法
UPDATE 视图名称 SET 列名=值 WHERE 条件;

-- 修改视图表中的城市名称北京为北京市
UPDATE city_country SET city_name='北京市' WHERE city_name='北京';

-- 查询视图
SELECT * FROM city_country;

-- 查询city表,北京也修改为了北京市
SELECT * FROM city;

-- 注意：视图表数据修改，会自动修改源表中的数据
```

- 修改视图表结构

```
-- 标准语法
ALTER VIEW 视图名称 [(列名列表)] AS 查询语句;

-- 查询视图2
SELECT * FROM city_country2;

-- 修改视图2的列名city_id为id
ALTER
VIEW
    city_country2 (id,city_name,cid,country_name)
AS
    SELECT t1.*,t2.country_name FROM city t1,country t2 WHERE t1.cid=t2.id;
```

7.视图的删除

- 删除视图

```
-- 标准语法
DROP VIEW [IF EXISTS] 视图名称;

-- 删除视图
DROP VIEW city_country;

-- 删除视图2，如果存在则删除
DROP VIEW IF EXISTS city_country2;
```

8.视图的总结

- 视图是一种虚拟存在的数据表
- 这个虚拟的表并不在数据库中实际存在
- 说白了，视图就是将一条SELECT查询语句的结果封装到了一个虚拟表中，所以我们在创建视图的时候，工作重心就要放在这条SELECT查询语句上
- 视图的好处
    - 简单
    - 安全
    - 数据独立

五、备份与还原

1.命令行方式

- 备份
    - 使用SecureCRT工具连接到Linux系统，输入：mysqldump -u root -p 数据库名称 > 文件保存路径

```
[root@itheima ~]# mysqldump -u root -p db1 > /root/mysql/db1.sql
Enter password:
```

    - 进入文件保存路径，查看文件是否存在

```
[root@itheima ~]# cd mysql
```

```
[root@itheima mysql]# ls
db1.sql                                  MySQL-server-5.6.22-1.el6.i686.rpm
MySQL-client-5.6.22-1.el6.i686.rpm       MySQL-shared-5.6.22-1.el6.i686.rpm
MySQL-devel-5.6.22-1.el6.i686.rpm        MySQL-shared-compat-5.6.22-1.el6.i686.rpm
MySQL-embedded-5.6.22-1.el6.i686.rpm     MySQL-test-5.6.22-1.el6.i686.rpm
```

- 恢复
  - 登录mysql数据库

```
[root@itheima mysql]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

  - 删除已经备份的数据库

```
mysql> drop database db1;
Query OK, 0 rows affected (0.03 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| db3                |
| db4                |
| mysql              |
| performance_schema |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

  - 重新创建名称相同的数据库

```
mysql> create database db1;
Query OK, 1 row affected (0.02 sec)
```

  - 使用该数据库

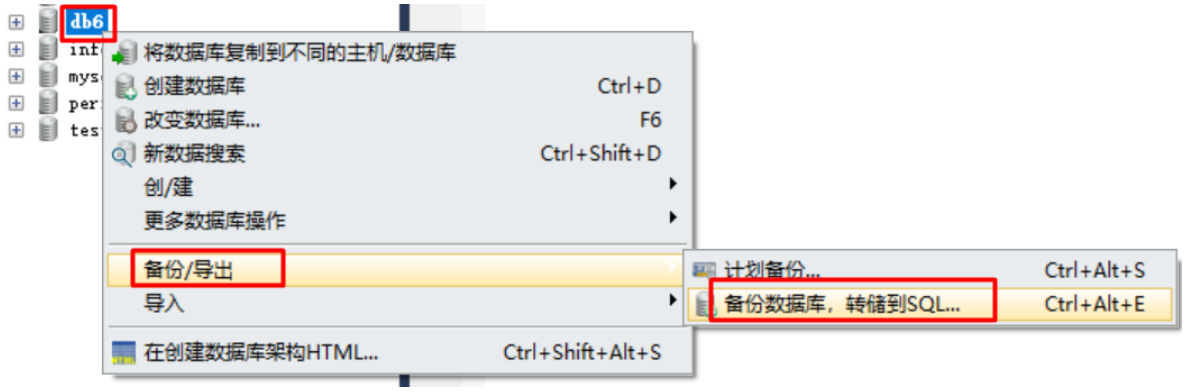
```
mysql> use db1;
Database changed
```

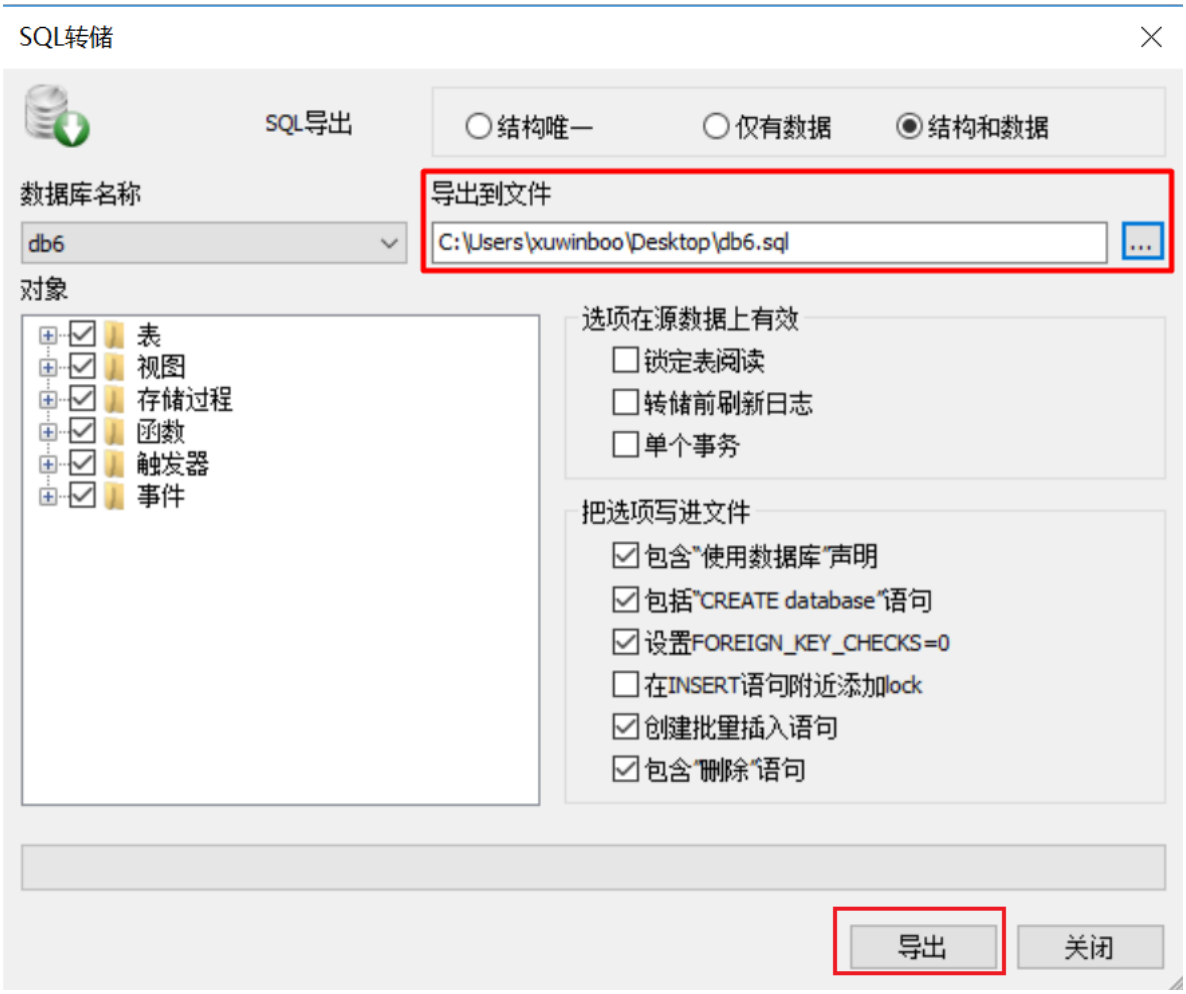
  - 导入文件执行：source 备份文件路径;

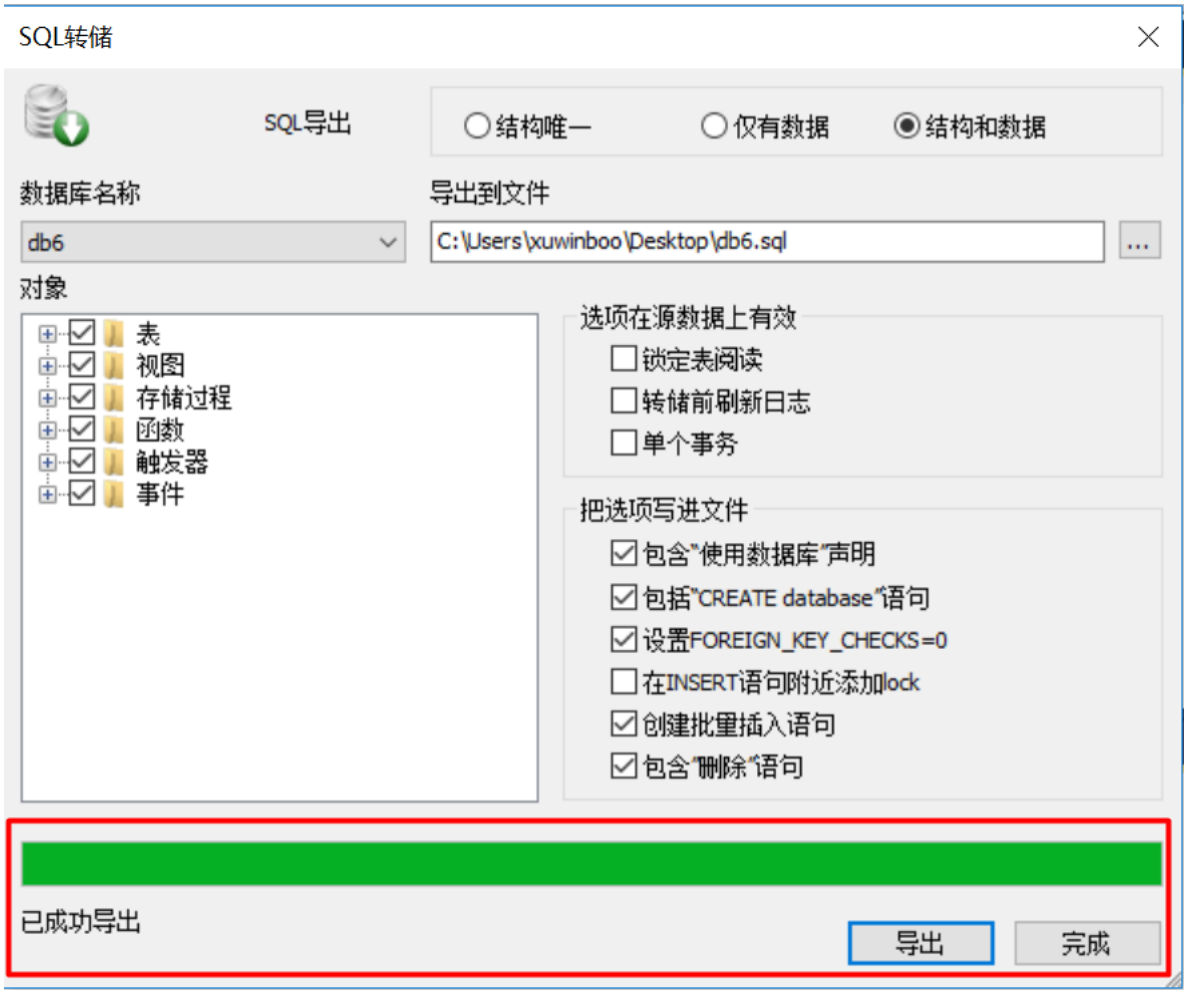
```
mysql> source /root/mysql/db1.sql;
Query OK, 0 rows affected (0.02 sec)
```

2.图形化界面方式

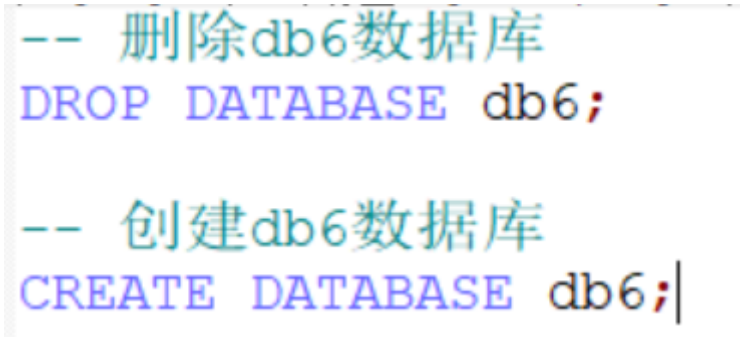
- 备份

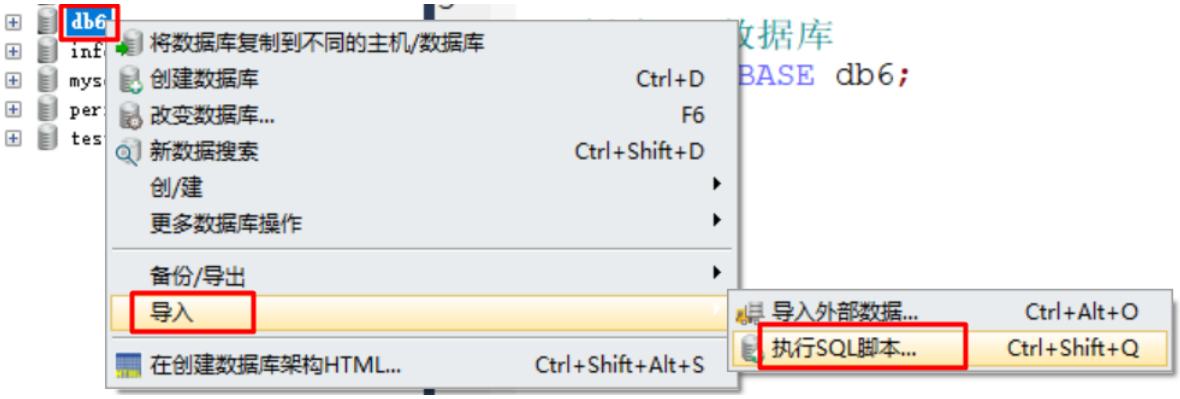

db6
将数据库复制到不同的主机/数据库
创建数据库 Ctrl+D
改变数据库... F6
新数据搜索 Ctrl+Shift+D
创/建
更多数据库操作
备份/导出
导入
在创建数据库架构HTML... Ctrl+Shift+Alt+S
计划备份... Ctrl+Alt+S
备份数据库，转储到SQL... Ctrl+Alt+E

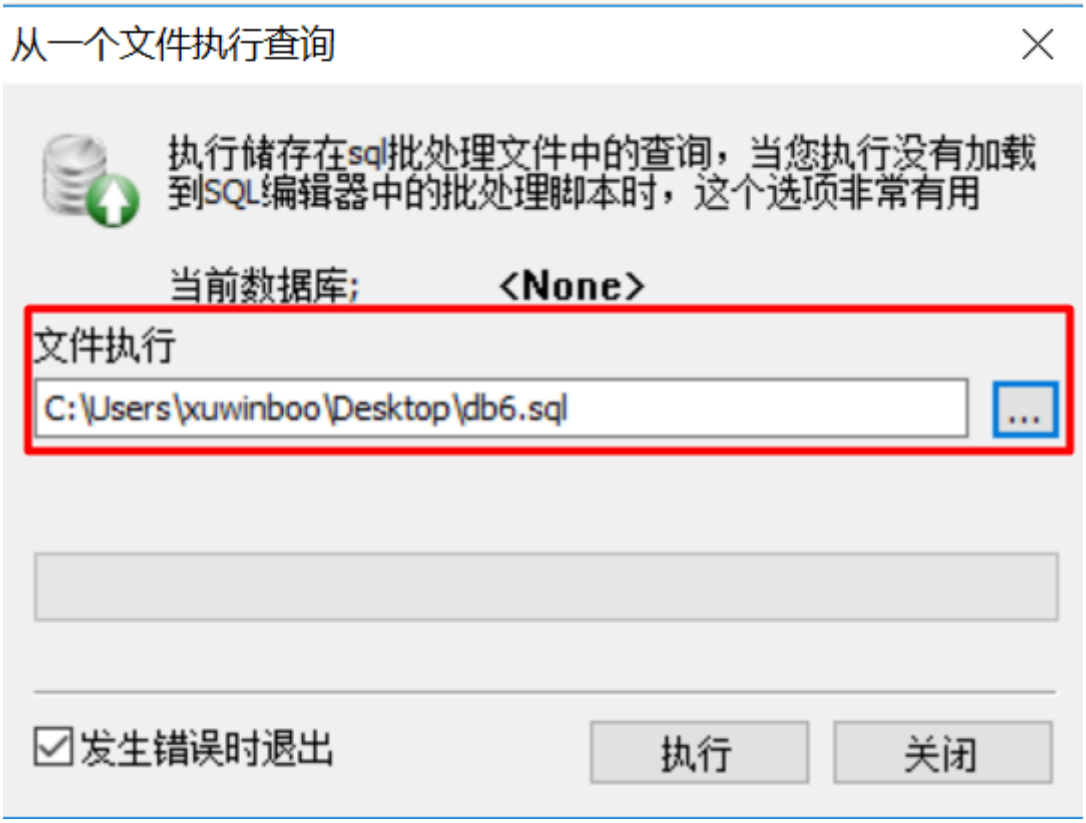

SQL转储
SQL导出
结构唯一
仅有数据
结构和数据
数据库名称
db6
导出到文件
C:\Users\xuwinboo\Desktop\db6.sql
对象
表
视图
存储过程
函数
触发器
事件
选项在源数据上有效
锁定表阅读
转储前刷新日志
单个事务
把选项写进文件
包含"使用数据库"声明
包括"CREATE database"语句
设置FOREIGN_KEY_CHECKS=0
在INSERT语句附近添加lock
创建批量插入语句
包含"删除"语句
导出
关闭

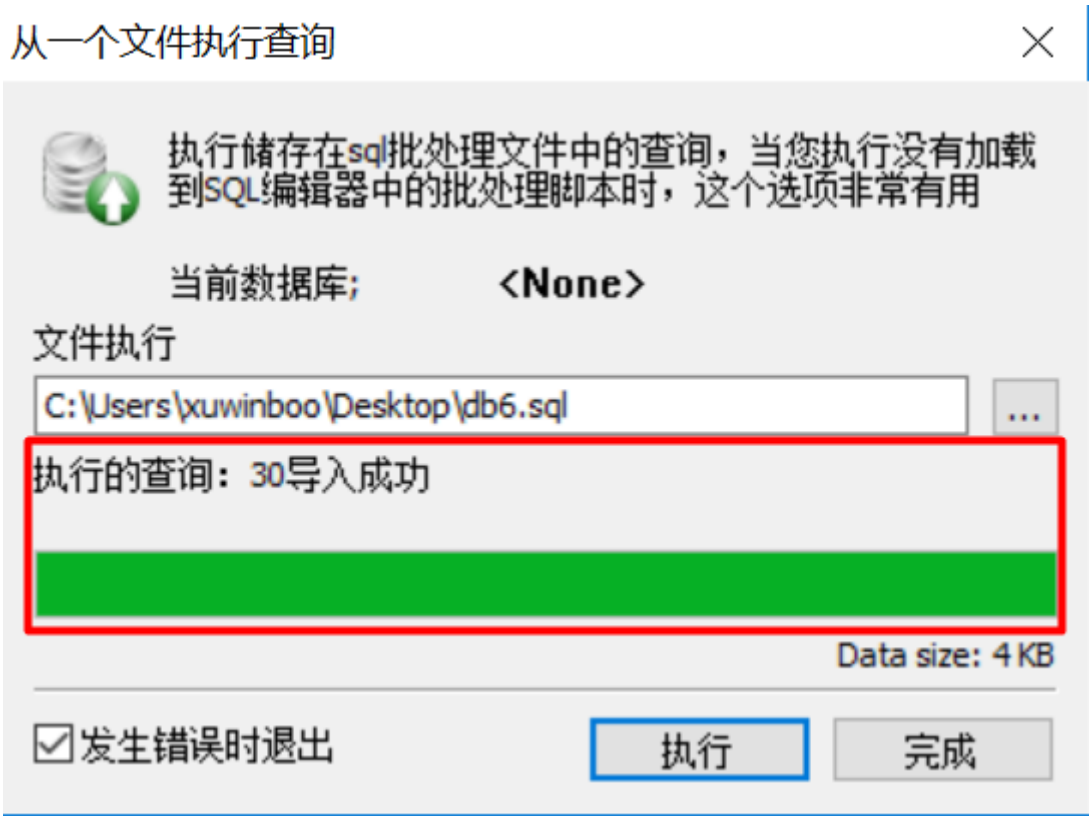

SQL转储
SQL导出
结构唯一
仅有数据
结构和数据
数据库名称
db6
导出到文件
C:\Users\xuwinboo\Desktop\db6.sql
对象
表
视图
存储过程
函数
触发器
事件
选项在源数据上有效
锁定表阅读
转储前刷新日志
单个事务
把选项写进文件
包含"使用数据库"声明
包括"CREATE database"语句
设置FOREIGN_KEY_CHECKS=0
在INSERT语句附近添加lock
创建批量插入语句
包含"删除"语句
已成功导出
导出
完成

- 恢复


-- 删除db6数据库
DROP DATABASE db6;
-- 创建db6数据库
CREATE DATABASE db6;


db6
将数据库复制到不同的主机/数据库
创建数据库 Ctrl+D
改变数据库... F6
新数据搜索 Ctrl+Shift+D
创/建
更多数据库操作
备份/导出
导入
在创建数据库架构HTML... Ctrl+Shift+Alt+S
导入外部数据... Ctrl+Alt+O
执行SQL脚本... Ctrl+Shift+Q


从一个文件执行查询
执行储存在sql批处理文件中的查询，当您执行没有加载到SQL编辑器中的批处理脚本时，这个选项非常有用
当前数据库;
<None>
文件执行
C:\Users\xuwinboo\Desktop\db6.sql
...
发生错误时退出
执行
关闭


从一个文件执行查询
执行储存在sql批处理文件中的查询，当您执行没有加载到SQL编辑器中的批处理脚本时，这个选项非常有用
当前数据库;
<None>
文件执行
C:\Users\xuwinboo\Desktop\db6.sql
...
执行的查询：30导入成功
Data size: 4 KB
发生错误时退出
执行
完成