Mysql

功能	SQL
查看所有的数据库	show databases;
创建数据库	create database [if not exists] mydb1 [charset=utf8]
切换 (选择要操作的) 数据库	use mydb1;
删除数据库	drop database [if exists] mydb1;
修改数据库编码	alter database mydb1 character set utf8;

--，#表示注释

ddl:对数据库的常用操作，对表结构的常用操作，修改表结构

```
-- 1.DDL操作之数据库操作
-- 查看所有数据库
show DATABASEs;
-- 创建数据库
CREATE DATABASE if not EXISTS mydb1;
-- 选择使用哪一个数据库
use mydb1;
-- 删除数据库
drop database if exists mydb1;
```

创建表是构建一张空表，指定这个表的名字，这个表有几列，每一列叫什么名字，以及每一列存储的数据类型。

```
CREATE DATABASE if not exists mydb1;
use mydb1;
create table if not exists student(
    sid int,
    name varchar(20),
    gender varchar(20),
    age int,
    birth date,
    address varchar(20)
);
```

数值类型

日期和时间类型

字符串类型

- 数值类型

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 byte	(-128，127)	(0，255)	小整数值
SMALLINT	2 bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 bytes	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 bytes	(-3.402 823 466 E+38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 bytes	(-1.797 693 134 862 315 7 E+308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL		依赖于M和D的值	依赖于M和D的值	小数值

字符串类型

类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串
TINYBLOB	0-255 bytes	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

日期类型

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 2147483647 秒，北京时间 2038-1-19 11:14:07，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

```
-- 查看当前数据库所有的表
show tables;
-- 查看指定表的创建语句
show create table student;
-- 查看表结构
desc student;
-- 删除表
drop table student;
```

dml：是指数据操作语言，对数据库中表的数据记录进行更新

插入insert，删除delete，更新update

insert into 表 (列名1,列名2,列名3...) **values (**值**1**,值**2**,值**3...**); //**向表**中**插入某些

insert into 表 **values (**值**1**,值**2**,值**3...**);** //向表**中**插入所有列

```
-- 数据插入
INSERT into student(sid,name,gender,age ,birth,address)
values(1002,'张三','男',18,'2001-12-23','北京'),
(1003,'张三','男',18,'2001-12-23','北京')
;
INSERT into student(sid) VALUES(1004);
INSERT into student values
(1005,'张三','男',18,'2001-12-23','北京');
```

update 表名 **set** 字段名**=**值,**字段名**=值...;**

update 表名 **set** 字段名**=**值,**字段名**=值...** **where** 条件;

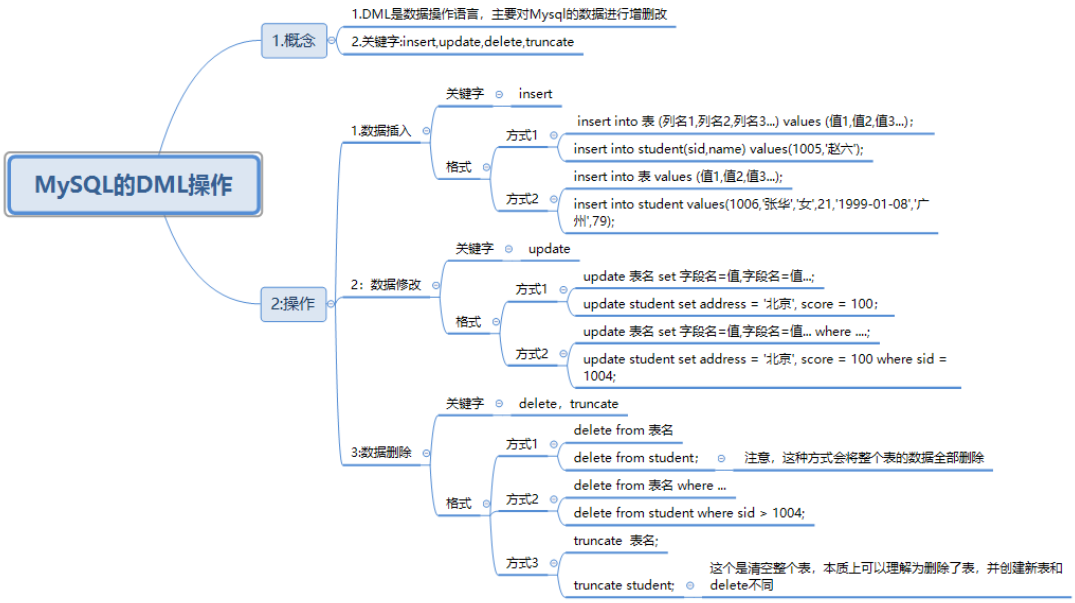
```
-- 数据修改
UPDATE student set address='重庆';
UPDATE student set address='北京'WHERE sid=1004;
UPDATE student set address='北京',age=20 WHERE sid=1005;
```

delete from 表名 **[**where** 条件];**

truncate table 表名 或者 **truncate** 表名

delete和truncate原理不同，delete只删除内容，而truncate类似于drop table ，可以理解为是将整个表删除，然后再创建该表；

```
-- 数据删除
delete from student where sid=1004;
delete from student;
truncate table student;
```




```
-- 1.创建表
/*创建员工表employee,id(员工编号)，name,gender,salary*/
CREATE TABLE if not EXISTS employee(
    id int,
    name varchar(20),
    gender varchar(20),
    salary int
);

-- 2.插入数据
INSERT into employee VALUES(1,'张三','男',2000),
(2,'李四','男',1000),(3,'王五','女',4000);
/*
1.'张三','男',2000
2.'李四','男',1000
3.'王五','女',4000
*/

-- 修改表数据
-- 3.1将所有员工薪水修改未5000元
UPDATE employee set salary=5000;

-- 3.2将姓名为张三的员工薪水修改为3000元
UPDATE employee set salary=3000 WHERE id=1;

-- 3.3将姓名为李四的员工薪水修改为4000元，gender改为女
UPDATE employee set salary=4000,gender='女'WHERE id=2;

-- 3.4将王五的薪水在原有基础上增加1000元
```

```
UPDATE employee set salary=salary+1000 WHERE id=3;
```

id	name	gender	salary
1	张三	男	3000
2	李四	女	4000
3	王五	女	6000

约束：表中数据的限制条件

表在设计的时候加入约束的目的就是为了保证表中的记录完整性和有效性，比如用户表有些列的值（手机号）不能为空，有些列的值（身份证号）不能重复。

主键约束，**自增长约束**，非空约束，唯一性约束，默认约束，零填充约束，外键约束

主键约束

MySQL主键约束是一个列或者多个列的组合，其值能**唯一地标识表中的每一行,**方便在RDBMS中尽快的找到某一行。

主键约束相当于 唯一约束 + 非空约束 的组合，**主键约束列不允许重复，也不允许出现空值。**

每个表**最多只允许一个主键。**

主键约束的关键字是：**primary key。**

当创建主键的约束时，系统默认会在所在的列和列组合上建立对应的唯一索引。

```
use mydb1;
create table emp1(
    eid int primary key,
    name VARCHAR(20),
    deptId int,
    salary double
);
create table emp2(
    eid INT,
    name VARCHAR(20),
    deptId INT,
    salary double,
    primary key(eid)
 );
--  主键作用
insert into emp2(eid,name, deptId,salary)
 values(1001,'张三',10,5000);
 INSERT into emp2 VALUES(1001,'李四',20,6000);
```

```
> 1062 - Duplicate entry '1001' for key 'emp2.PRIMARY'
```

```
 INSERT into emp2 VALUES(1002,'李四',20,6000);
```

主键约束的列非空而且唯一

联合主键：

1. 当主键是由多个字段组成时，不能直接在字段名后面声明主键约束。

2. 一张表只能有一个主键，联合主键也是一个主键。

create table 表名**(**

...

primary key (字段1，字段2，**...,**字段n**)**

);

```
create table emp3(
  name varchar(20),
  deptId int,
  salary double,
  CONSTRAINT pk2 primary key(name,deptId)
);
```

（只要不是完全一样就能插入）

（必须各列都不为空才能插入）

通过修改表结构添加主键：

create table 表名**(**

...

);

alter table <表名> **add primary key**（**字段列表**);

删除主键：**alter table** <数据表名> **drop primary key**;

自增长约束(auto_increment)

当主键定义为自增长后，这个主键的值就不再需要用户输入数据了，而由数据库系统根据定义自动赋值，每增加一条记录，主键会自动以相同的步长进行增长

字段名 数据类型 auto_increment

```
-- 自增长约束
CREATE table t_user1(
    id int PRIMARY key auto_increment,
    name varchar(20)
);
INSERT into t_user1 (name)values ('zhangsan');
```

默认情况下，auto_increment的初始值是 1，每新增一条记录，字段值自动加 1。
一个表中只能有一个字段使用 auto_increment约束，且该字段必须有唯一索引，以避免序号重复（即为主键或主键的一部分）。
auto_increment约束的字段必须具备 NOT NULL 属性。
auto_increment约束的字段只能是整数类型（TINYINT、SMALLINT、INT、BIGINT 等。
auto_increment约束字段的最大值受该字段的数据类型约束，如果达到上限，auto_increment就会失效。

指定自增字段初始值：如果第一条记录设置了该字段的初始值，那么新增加的记录就从这个初始值开始自增。例如，如果表中插入的第一条记录的 id 值设置为 5，那么再插入记录时，id 值就会从 5 开始往上增加

创建表时：

```
create table t_user2 (
  id int primary key auto_increment,
  name varchar(20)
)auto_increment=100;
```

创建表后：

```
create table t_user3 (
  id int primary key auto_increment,
  name varchar(20)
);
alter table t_user2 auto_increment=100;
```

delete数据之后自动增长从断点开始

truncate数据之后自增长从默认起始值开始

非空约束（not null）

方式1：<字段名><数据类型> not null;

方式2：alter table 表名 modify 字段 类型 not null;

```
CREATE table t_user2(
    id int,
    name varchar(20) not null,
    adress varchar(20)
);
```

```
alter table t_user7 modify name varchar(20) not null;
alter table t_user7 modify address varchar(20) not null;
```

删除非空约束：

```
alter table t_user7 modify name varchar(20) ;
alter table t_user7 modify address varchar(20) ;
```

唯一约束（unique）

方式1：<字段名> <数据类型> unique

方式2： alter table 表名 add constraint 约束名 unique(列);

```
alter table t_user9 add constraint unique_ph unique(phone_number);
```

```
-- alter table <表名> drop index <唯一约束名>;
alter table t_user9 drop index unique_ph;
```

默认约束（default）

方式1： <字段名> <数据类型> default <默认值>;

方式2: alter table 表名 modify 列名 类型 default 默认值;

删除：alter table <表名> **modify** column <字段名> <类型> default null;

DQL：基本查询

```
select
  [all|distinct]
  <目标列的表达式1> [别名],
  <目标列的表达式2> [别名]...
from <表名或视图名> [别名],<表名或视图名> [别名]...
[where<条件表达式>]
[group by <列名>
[having <条件表达式>]]
[order by <列名> [asc|desc]]
[limit <数字或者列表>];
```

select *| 列名 from 表 where 条件

```
-- 创建数据库
create database if not exists mydb2;
use mydb2;
-- 创建商品表:
create table product(
 pid int primary key auto_increment, -- 商品编号
 pname varchar(20) not null , -- 商品名字
 price double,  -- 商品价格
 category_id varchar(20) -- 商品所属分类
);
insert into product values(null,'海尔洗衣机',5000,'c001');
insert into product values(null,'美的冰箱',3000,'c001');
insert into product values(null,'格力空调',5000,'c001');
insert into product values(null,'九阳电饭煲',200,'c001');
insert into product values(null,'啄木鸟衬衣',300,'c002');
insert into product values(null,'恒源祥西裤',800,'c002');
insert into product values(null,'花花公子夹克',440,'c002');
insert into product values(null,'劲霸休闲裤',266,'c002');
insert into product values(null,'海澜之家卫衣',180,'c002');
insert into product values(null,'杰克琼斯运动裤',430,'c002');
insert into product values(null,'兰蔻面霜',300,'c003');
insert into product values(null,'雅诗兰黛精华水',200,'c003');
insert into product values(null,'香奈儿香水',350,'c003');
insert into product values(null,'SK-II神仙水',350,'c003');
insert into product values(null,'资生堂粉底液',180,'c003');
insert into product values(null,'老北京方便面',56,'c004');
insert into product values(null,'良品铺子海带丝',17,'c004');
insert into product values(null,'三只松鼠坚果',88,null);

-- 1.查询所有的商品.
SELECT pid,pname,price,category_id from product ;
SELECT * from product;
-- 2.查询商品名和商品价格.
SELECT   pname,price from product;
-- 3.别名查询.使用的关键字是as（as可以省略的）.
-- 3.1表别名:
SELECT * from product as p;
SELECT * from product p;
-- 3.2列别名:
```

```sql
SELECT pname as '商品名', price'商品价格' from product;
-- 4.去掉重复值.
SELECT distinct price from product;
-- 5.查询结果是表达式（运算查询）：将所有商品的价格+10元进行显示.
SELECT pname, price+10 new_price from product;
```

运算符

4种运算符

算数运算符

算术运算符	说明
+	加法运算
-	减法运算
*	乘法运算
/ 或 DIV	除法运算，返回商
% 或 MOD	求余运算，返回余数

比较运算符

比较运算符	说明
=	等于
< 和 <=	小于和小于等于
> 和 >=	大于和大于等于
<=>	安全的等于，两个操作码均为NULL时，其所得值为1；而当一个操作码为NULL时，其所得值为0
<> 或!=	不等于
IS NULL 或 ISNULL	判断一个值是否为 NULL
IS NOT NULL	判断一个值是否不为 NULL
LEAST	当有两个或多个参数时，返回最小值
GREATEST	当有两个或多个参数时，返回最大值
BETWEEN AND	判断一个值是否落在两个值之间
IN	判断一个值是IN列表中的任意一个值
NOT IN	判断一个值不是IN列表中的任意一个值
LIKE	通配符匹配
REGEXP	正则表达式匹配

逻辑运算符

逻辑运算符	说明
NOT 或者 !	逻辑非
AND 或者 &&	逻辑与
OR 或者 ||	逻辑或
XOR	逻辑异或

```sql
-- 查询商品名称为“海尔洗衣机”的商品所有信息：
SELECT * from product where pname='海尔洗衣机';
-- 查询价格为800商品
SELECT * from product where price=800;
-- 查询价格不是800的所有商品
SELECT * from product where price!=800;
SELECT * from product where price<>800;
SELECT * from product where not(price=800);
-- 查询商品价格大于60元的所有商品信息
SELECT * from product where price>=60;
-- 查询商品价格在200到1000之间所有商品
SELECT * from product where price between 200 and 1000;
SELECT * from product where price>=200 and price <=1000;
-- 查询商品价格是200或800的所有商品
SELECT * from product where price in(200,800);
SELECT * from product where price=200 or price =800;
SELECT * from product where price=200 ||price =800;
-- 查询含有‘裤'字的所有商品
SELECT * from product where pname like '%裤';
-- 查询以'海'开头的所有商品
SELECT * from product where pname like '海%';
-- 查询第二个字为'蔻'的所有商品
SELECT * from product where pname like '_蔻%';
-- 查询category_id为null的商品
SELECT * from product where category_id is null;
-- 查询category_id不为null分类的商品
SELECT * from product where category_id is not null;
-- 使用least求最小值
SELECT least(10,20,30) as small_number;
SELECT least(10,null,20);
-- 使用greatest求最大值
SELECT least(10,20,30) as big_number;
SELECT least(10,null,20);
```

排序查询

select

字段名1，字段名2，……

from 表名

order by 字段名1 [asc|desc]，字段名2[asc|desc]……

```sql
-- 1.使用价格排序(降序)
SELECT * from product order by price asc;
SELECT * from product order by price desc;
-- 2.在价格排序(降序)的基础上，以分类排序(降序)
SELECT * from product order by price desc,category_id desc;
-- 3.显示商品的价格(去重复)，并排序(降序)
SELECT DISTINCT price from product order by price desc;
```

聚合查询

聚合函数	作用
count()	统计指定列不为NULL的记录行数；
sum()	计算指定列的数值和，如果指定列类型不是数值类型，那么计算结果为0
max()	计算指定列的最大值，如果指定列是字符串类型，那么使用字符串排序运算；
min()	计算指定列的最小值，如果指定列是字符串类型，那么使用字符串排序运算；
avg()	计算指定列的平均值，如果指定列类型不是数值类型，那么计算结果为0

```
-- 1 查询商品的总条数
SELECT count(pid) from product;
SELECT count(*) from product;
-- 2 查询价格大于200商品的总条数
SELECT count(*) from product where price >200;
-- 3 查询分类为'c001'的所有商品的总和
SELECT sum(price) from product where category_id='c001';
-- 4 查询商品的最大价格
SELECT max(price) from product;
-- 5 查询商品的最小价格
SELECT min(price) from product;
SELECT max(price) ,min(price) from product;
-- 6 查询分类为'c002'所有商品的平均价格
SELECT avg(price) from product where category_id='c002';
```

1、count函数对null值的处理

如果count函数的参数为星号（*），则统计所有记录的个数。而如果参数为某字段，不统计含null值的记录个数。

2、sum和avg函数对null值的处理

这两个函数忽略null值的存在，就好象该条记录不存在一样。

3、max和min函数对null值的处理

max和min两个函数同样忽略null值的存在。

分组查询

select 字段1,字段2... from 表名 group by 分组字段 having 分组条件;

如果要进行分组的话，则SELECT子句之后，只能出现分组的字段和统计函数，其他的字段不能出现

分组之后的条件筛选-having

select 字段1,字段2... from 表名 group by 分组字段 having 分组条件;

select category_id ,count*() from product group by category_id having count()* > 1;

SQL:from->group by -> count(pid)->select->having->order by

分页查询

-- 方式1-显示前n条

select 字段1，字段2... from 表明 limit n

-- 方式2-分页显示

select 字段1，字段2... from 表明 limit m,n

m: 整数，表示从第几条索引开始，计算方式 （当前页-1）*每页显示条数

n: 整数，表示查询多少条数据

insert into select语句：将一张表的数据导入到另一张表中，可以使用INSERT INTO SELECT语句 。

insert into Table2(field1,field2,...) select value1,value2,... from Table1 或者：

insert into Table2 select * from Table1

正则表达式

模式	描述
^	匹配输入字符串的开始位置。
$	匹配输入字符串的结束位置。
.	匹配除 "\n" 之外的任何单个字符。
[...]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
[^...]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'。
p1|p2|p3	匹配 p1 或 p2 或 p3。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。

模式	描述
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,m}	m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。

```
-- ^ 在字符串开始处进行匹配
SELECT  'abc' REGEXP '^a';

-- $ 在字符串末尾开始匹配
SELECT  'abc' REGEXP 'a$';
SELECT  'abc' REGEXP 'c$';

-- . 匹配任意字符
SELECT  'abc' REGEXP '.b';
SELECT  'abc' REGEXP '.c';
SELECT  'abc' REGEXP 'a.';

-- [...] 匹配括号内的任意单个字符
SELECT  'abc' REGEXP '[xyz]';
SELECT  'abc' REGEXP '[xaz]';
-- [^...] 注意^符合只有在[]内才是取反的意思，在别的地方都是表示开始处匹配
SELECT  'a' REGEXP '[^abc]';
SELECT  'x' REGEXP '[^abc]';
SELECT  'abc' REGEXP '[^a]';

-- a* 匹配0个或多个a,包括空字符串。 可以作为占位符使用.有没有指定字符都可以匹配到数据
```

```
SELECT 'stab' REGEXP '.ta*b';
SELECT 'stb' REGEXP '.ta*b';
SELECT '' REGEXP 'a*';

-- a+  匹配1个或者多个a,但是不包括空字符
SELECT 'stab' REGEXP '.ta+b';
SELECT 'stb' REGEXP '.ta+b';
-- a?  匹配0个或者1个a
SELECT 'stb' REGEXP '.ta?b';
SELECT 'stab' REGEXP '.ta?b';
SELECT 'staab' REGEXP '.ta?b';

-- a1|a2  匹配a1或者a2，
SELECT 'a' REGEXP 'a|b';
SELECT 'b' REGEXP 'a|b';
SELECT 'b' REGEXP '^(a|b)';
SELECT 'a' REGEXP '^(a|b)';
SELECT 'c' REGEXP '^(a|b)';

-- a{m} 匹配m个a

SELECT 'auuuuc' REGEXP 'au{4}c';
SELECT 'auuuuc' REGEXP 'au{3}c';
-- a{m,n} 匹配m到n个a,包含m和n

SELECT 'auuuuc' REGEXP 'au{3,5}c';
SELECT 'auuuuc' REGEXP 'au{4,5}c';
SELECT 'auuuuc' REGEXP 'au{5,10}c';

-- (abc) abc作为一个序列匹配，不用括号括起来都是用单个字符去匹配，如果要把多个字符作为一个整体去匹配就需要用到括号，所以括号适合上面的所有情况。
SELECT 'xababy' REGEXP 'x(abab)y';
SELECT 'xababy' REGEXP 'x(ab)*y';
SELECT 'xababy' REGEXP 'x(ab){1,2}y';
```

多表操作

多表关系

一对一，一对多/多对一，多对多

外键约束

MySQL 外键约束（FOREIGN KEY）是表的一个特殊字段，经常与主键约束一起使用。对于两个具有关联关系的表而言，相关联字段中主键所在的表就是主表（父表），外键所在的表就是从表（子表）。

[constraint <外键名>] foreign key 字段名 [，字段名2，...] references <主表名> 主键列1 [，主键列2，...]

```
-- 创建部门表
create table if not exists dept2(
  deptno varchar(20) primary key ,  -- 部门号
  name varchar(20) -- 部门名字
);
-- 创建员工表
```

```
create table if not exists emp2(
  eid varchar(20) primary key , -- 员工编号
  ename varchar(20), -- 员工名字
  age int,  -- 员工年龄
  dept_id varchar(20)  -- 员工所属部门

);
-- 创建外键约束
alter table emp2 add constraint dept_id_fk foreign key(dept_id) references dept2
(deptno);
```

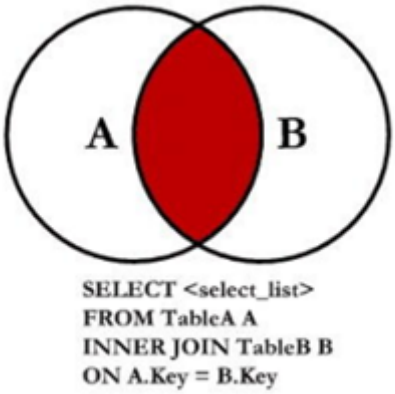
```
-- 1、添加主表数据
 -- 注意必须先给主表添加数据
insert into dept values('1001','研发部');
insert into dept values('1002','销售部');
insert into dept values('1003','财务部');
insert into dept values('1004','人事部');

-- 2、添加从表数据
  -- 注意给从表添加数据时，外键列的值不能随便写，必须依赖主表的主键列
insert into emp values('1','乔峰',20, '1001');
insert into emp values('2','段誉',21, '1001');
insert into emp values('3','虚竹',23, '1001');
insert into emp values('4','阿紫',18, '1002');
insert into emp values('5','扫地僧',35, '1002');
insert into emp values('6','李秋水',33, '1003');
insert into emp values('7','鸠摩智',50, '1003');
insert into emp values('8','天山童姥',60, '1005');  -- 不可以
-- 3、删除数据
 /*
   注意：
        1：主表的数据被从表依赖时，不能删除，否则可以删除
        2：从表的数据可以随便删除
 */
delete from dept where deptno = '1001'; -- 不可以删除
delete from dept where deptno = '1004'; -- 可以删除
delete from emp where eid = '7'; -- 可以删除
```

删除：**alter table <表名> drop foreign key <外键约束名>;**

多表联合查询

格式：select * from 表1,表2,表3....;

内连接查询



隐式内连接（SQL92标准）：select * from A,B where 条件;

显示内连接（SQL99标准）：select * from A inner join B on 条件;

```
-- 查询每个部门的所属员工
select * from dept3,emp3 where dept3.deptno = emp3.dept_id;
select * from dept3 inner join emp3 on dept3.deptno = emp3.dept_id;
-- 查询研发部和销售部的所属员工
select * from dept3,emp3 where dept3.deptno = emp3.dept_id and name in( '研发
部','销售部');
select * from dept3 join emp3 on dept3.deptno = emp3.dept_id and name in( '研发
部','销售部');

-- 查询每个部门的员工数,并升序排序
select deptno,count(1) as total_cnt from dept3,emp3 where dept3.deptno =
emp3.dept_id group by deptno order by total_cnt;

select deptno,count(1) as total_cnt from dept3 join emp3 on dept3.deptno =
emp3.dept_id group by deptno order by total_cnt;
-- 查询人数大于等于3的部门，并按照人数降序排序
select deptno,count(1) as total_cnt from dept3,emp3 where dept3.deptno =
emp3.dept_id group by deptno having total_cnt >= 3 order by total_cnt desc;

select deptno,count(1) as total_cnt from dept3 join emp3 on dept3.deptno =
emp3.dept_id group by deptno having total_cnt >= 3 order by total_cnt desc;
```

外连接查询

```
-- 外连接查询
-- 查询哪些部门有员工，哪些部门没有员工
use mydb3;
select * from dept3 left outer join emp3 on dept3.deptno = emp3.dept_id;

-- 查询哪些员工有对应的部门，哪些没有
select * from dept3 right outer join emp3 on dept3.deptno = emp3.dept_id;


-- 使用union关键字实现左外连接和右外连接的并集
select * from dept3 left outer join emp3 on dept3.deptno = emp3.dept_id
union
select * from dept3 right outer join emp3 on dept3.deptno = emp3.dept_id;
```

子查询（all，any，some，in，exists）

```
-- 查询年龄最大的员工信息，显示信息包含员工号、员工名字，员工年龄
select eid,ename,age from emp3 where age = (select max(age) from emp3);
-- 查询年研发部和销售部的员工信息，包含员工号、员工名字
select eid,ename,t.name from emp3 where dept_id in (select deptno,name from
dept3 where name = '研发部' or name = '销售部') ;
-- 查询研发部20岁以下的员工信息,包括员工号、员工名字，部门名字
select eid,age,ename,name from (select * from dept where name = '研发部 ')t1,
(select * from emp3 where age <20)t2
```

自关联查询：

select 字段列表 from 表1 a , 表1 b where 条件;或者 select 字段列表 from 表1 a [left] join 表1 b on 条件;

```
-- 创建表,并建立自关联约束
create table t_sanguo(
    eid int primary key ,
    ename varchar(20),
    manager_id int,
 foreign key (manager_id) references t_sanguo (eid)  -- 添加自关联约束
);
-- 添加数据
insert into t_sanguo values(1,'刘协',NULL);
insert into t_sanguo values(2,'刘备',1);
insert into t_sanguo values(3,'关羽',2);
insert into t_sanguo values(4,'张飞',2);
insert into t_sanguo values(5,'曹操',1);
insert into t_sanguo values(6,'许褚',5);
insert into t_sanguo values(7,'典韦',5);
insert into t_sanguo values(8,'孙权',1);
insert into t_sanguo values(9,'周瑜',8);
insert into t_sanguo values(10,'鲁肃',8);

-- 进行关联查询
-- 查询每个三国人物及他的上级信息，如:  关羽  刘备
select * from t_sanguo a, t_sanguo b where a.manager_id = b.eid;
```