```
#针对库的增删改查
   #增
   create database; 库名
   create database; 库名 charset='编码格式'
   #查
   show databases;
   show create database; 库名
   #改
   alter database 库名 charset='编码'; ##修改编码
   #删
   drop database; 库名

#针对表的增删改查
   操作表（文件）的时候需要指定所在的库
   #查看当前所在的库的名字
   select database();
   #切换库
   use 库名
   #增
      #创建表
      create table 表名(id int,name char(n)) engine=引擎名;
         ^创建一张表格，有两个字段id和name，指定id必须是
         数字，name一定要是字符串，指定字符串长度为n
      #查表
      show tables;
      show create table 表名;
      describe 表名; 简写：desc 表名
         ^查看表的信息
      #改表
      alter table 表名 modify name char(n); #修改表的名字长度
      #删表
      drop table 表名

#可以用绝对路径进行操作

#针对数据的增删改查
      #增
      insert into 表名 values(1,'jason');
         ^括号里面的两个数据对应两个字段
         ^可以采用位定位的方式
            insert into 表名(字段2，字段1) values（字段2的数据，字段1的数据）;
      insert into 表名 values(可以多条);
      #查
      select * from 表名; 将表所有数据展示 #数据集较大不建议使用
      select 字段 from 表名; #选取一个字段查看
      #改
      update 表名 set 字段='某个数据名字' where 字段条件;
         ^update t1 set name='DSB' where id>1;
      #删
      delete from 表名 where 条件;
         ^delete from t1 where name='DSB'
         #将表所有数据清空
         delete from 表名;
```

#存储引擎
日常生活中，文件格式很多
存储引擎针对不同的文件格式会有对应的存储处理机制

#innodb
^mysql 5.5版本及之后的默认存储引擎
^支持行锁，外键，事务
^(row-level-Locking,transactions,foreign keys);
^存储更加安全
#myisam
^mysql 5.5版本之前的默认存储引擎
^速度更快，但是由于主要注重数据安全，已经切换
#memory
^内存引擎（数据都在内存，断电数据丢失）
#blackhole
^无论存什么，都立刻消失（黑洞）

#查看所有的所有存储引擎
show engines;
#不同存储引擎在存储表的时候 异同点

##
#创建表的完整语法
create table 表名(
字段1 类型(宽度) 约束条件,
字段2 类型(宽度) 约束条件,
字段3 类型(宽度) 约束条件
)
^同一张表中，字段名不能重复
^宽度和约束条件是可选的，字段名和字段类型是必须的
^约束条件写的话可以写多个
字段1 类型(宽度) 约束条件1,约束条件2，...
^最后一行不能有逗号
^宽度：一般情况下指的是对存储数据的限制,默认宽度为1
^只有整型数据里买的数字不是表示限制宽度，而是显示长度
id in(8)
^如果数字没有超出八位，默认用空格填充至八位
^如果达到八位，存入实际的值（遵循数据类型最大范围）

#修改表的完整语法（了解）
修改表名
alter table 表名 rename 新表面；
增加字段
alter table add 字段名 字段类型（宽度） 约束条件; #不写默认加在表最后
alter table add 字段名 字段类型（宽度） 约束条件 first; #直接添加在表最前面一列
alter table add 字段名 字段类型（宽度） 约束条件 after 字段名; #添加在某一字段后面
删除字段
alter table 表名 dorp 字段名;
修改字段
alter table 表名 modify 字段名 字段类型（宽度） 约束条件;
alter table 表名 change 旧字段名 新字段名 字段类型（宽度） 约束条件;

#复制表格（了解）
create table 新表名 select * from 被复制的表 复制条件;
^只能复制表象数据，但是不能复制主键，外键，索引等

```
#############################################################################################
#数据类型
  #整形（主要内容见表格）
    tinyint     1字节
    smallint   2字节
    mediumint  3字节
    int          4字节
    bigint
    ^存储年龄，等级，id，号码...不同内容使用不同的存储类型
    ^默认带符号，如果有符号，符号占一位
    ^存储数据超出范围会存储最接近的值
    ^针对整型字段，括号内无需指定宽度，因为它默认的宽度
       已经足够显示足够的数据了

  #浮点类型
    float       #总共255位，小数占30位
    double    #总共255位，小数占30位
    decimal      #总共65位，小数占30位
    精确度：float< double< decimal

  #字符类型
    char
      ^定长
        ^char(4) 超过四个字符，直接报错，不够四个空格补全
        ^存取简单，直接按照固定字符存取数据
    varchar
      ^变长
        ^varchar(4) 数据超过四个直接报错，不够时不补全
        ^节省空间
        ^存取麻烦
          ^存储的时候添加数据长度报头
            取的时候先取报头，再按着报头取数据

    现在用的较多的是varchar

    #介绍小方法： char_length 统计字段长度
    select char_length from 表名
      ^char硬盘上存储的数据如果补充空格
        显示时Mysql会自动将多余空格剔除显示
        利用 set global sql_mode='PAD_CHAR_TO_FULL_LENGTH';

  #时间类型
    date     年月日
    datetime  年月日时分秒（多time就多时分秒）
    time     时分秒
    year     1970年以后才可以输入

  #枚举和集合类型
    枚举（enum）  多选一
      ^后期存储数据enum字段只能从当初加入的选项中选一个存储
    集合（set）      多选多
      ^后期存储数据set字段只能从当初加入的选项中选一个或多个存储
#############################################################################################
```

```
#严格模式
   查看严格模式
   show variable like "%mode";
      ^5.6版本没有任何严格模式
   修改严格模式
   set session      只在当前窗口有效，临时修改
   set global       全局有效
      set global sql_mode = 'STRICT_TRANS_TABLES';
         ^设置之后，重新进入服务端生效

#模糊查询/匹配
   关键字 like
      %  匹配任意多个字符
      _  匹配任意单个字符




###以后创建表的id（数据的唯一标识）字段的时候###
   id int primary key auto_increment
#################################################
   ###^自增的主键在删除表中的数据之后，再次创建数据，自增不会停止，会延续上次的自增数
   ###^想要完全删除数据和重置自增主键，使用命令：truncate 表名
#########################################################################################


#########################################################################################
#建立表格时的约束条件
   not NULL，使存储不能存NULL
            create table t8(id int,name char not null);

   unsigned,加入后使得数字数据无符号

   zerofill,用0填充（原本为空格填充了）

   default默认值
      create table t1(
         id int,
         name char(16),
         gender enum('male','female') default 'male'
         );

   unique
      单一唯一
         create table t(
         id int,
         name char(36) unique)
      联合唯一
         create table t(
         id int,
         ip char(32),
         port int,
         unique(ip,port));

   primary key 主键
```

^单从约束效果来看，primary key = not null + unique
^非空且唯一
^单个字段主键
create table t(id int primary key);
insert into t values(null); 报错
insert into t values(1),(1); 报错
insert into t values(1),(2);

^联合主键（多个字段联合做一个主键，用的不多）
create table t(
id int,
ip char(32),
port int,
primary key(ip,port));

^除了有约束的效果之外，他还是innodb存储引擎组织数据的依据
因为他类似存储引擎的目录，能够帮助提示查询效率并且也是依据
^一张表中有且只有一个主键，如果没有设置主键
那么他会从上往下搜索直到遇到第一个非空且唯一的字段将他自动升级为主键
create table t(
id int,
name char(16),
age int not null unique,
addr char(32) not null unique
) #此时age会自动升级为主键

^如果表中没有主键也没有其他任何的非空唯一字段
那么innodb会采用内部提供的一个隐藏字段作为主键，看不到，也无法使用

^一张表中通常哦都应该有一个主键字段，并且通常将id字段作为主键
create table t(
id int primary key,

name char(16));

auto_increment自增
^通常都是加在主键上的，不能给普通字段加
#当编号很多的时候，人为维护太麻烦
create table t(
id int primary key auto_increment,
name varchar);
insert into t(name) values('gh','lsm'); 编号会自增

#数据操作的约束条件
#聚合函数
max()
min()
avg()
count()
#where 约束条件
查询id大于等于三小于等于6的数据
select id,name,age from emp where id>=3 and id<=6;
select id,name,age from emp where id between 3 and 6; 这里的between是闭区间

查询薪资是17000或者18000的数据
select * from emp where salary=17000 or salary=18000;
salary * from emp where salary in (17000,18000);

```
查询员工姓名中包含字母O的员工的姓名和薪资
  select name,salary from emp where name like '%O%'

查询员工姓名是四个字符的姓名和薪资
  select name,salary from emp where name like '____';
  select name,salary from emp where char_length(name)=4;

查询id小于3或者id大于6的数据
  select name from emp where id not between 3 and 6;

查询薪资不在17000，18000范围的数据
  select * from emp where salary not in(17000,18000);

查询岗位描述为空的员工姓名和岗位名 针对NULL不用符号，用is
  select name,post from emp where post_comment is null;

#group by 以...分组
  ^男女比例，部门平均薪资，国家之间数据统计
  ^分组之后最小可操作单位应该是组，而不是组内单个数据
  ^关键字where 和 group by同时出现的时候 group by必须在where的后面
    ^where实现对整体数据进行过滤之后在进行分组
  ^聚合函数只能在分组之后使用，where 筛选条件中不能使用聚合函数
  ^不分组，整张表为一组

  按照部门分组
    select * from emp group by post;
      ^没有设置严格模式是可以正常执行的，返回是分组之后每个组的第一个数据
        不符合分组规范：分组之后不应该考虑 单个数据，而应该考虑组为操作单位
        分组之后没有办法获取组内单个数据
      ^如果设置了严格模式，上述命令会直接报错

    select post from emp group by post;
      ^按什么分组就只能拿到分组，其他字段不能直接获取
        需要借助一些方法(聚合函数)

    获取每个部门的最高薪资
      select post,max(salary) from emp group by post;
      select post as '部门',max(salary) as '最高薪资' from emp group by post;
      select post '部门',max(salary) '最高薪资' from emp group by post;
        ^as可以给字段取别名，也可以直接省略不写，但是不推推荐省略

    #####as可以给字段取别名，也可以给表格取别名####
      select emp.id,emp.name from emp;
      select emp.id,emp.name from emp as t1;  报错
        ^当前临时有效，此时emp表名暂时变为t1

    获取每个部门的最低薪资
      select post,min(salary) from emp group by post;

    获取每个部门的平均薪资
      select post,avg(salary) from emp group by post;

    获取每个部门的薪资总和
      select post,sum(salary) from emp group by post;

    获取每个部门的人数
      ^不能count为null的数据
      select post,count(id) from emp group by post;  常用
```

```
        select post,count(salary) from emp group by post;
        select post,count(age) from emp group by post;
        select post,count(非null数据) from emp group by post;

    group_concat()
      ^获取分组后的其他字段所有的值，还支持拼接操作
      查询分组后的部门名称及每个部门下所有员工的姓名
        select post,group_comcat(name) from emp group by post;
        select post,group_comcat(name,'拼接在name后面的字符') from emp group by post;
        select post,group_comcat(name,':',salary) from emp group by post;

    concat()
      ^不分组的时候用的
        select concat('NAME',name),concat('SAL',salary) from emp



      统计各部门年龄在30岁以上的员工的平均薪资
        1.select * from emp where age>30;
        2.select * from emp where age>30 group by post;
        3.select post,avg(salary) from emp where age>30 group by post;

#having 筛选
  ^用于分组之后的筛选条件
  ^语法和where 一致
  ^即可以直接使用聚合函数
    统计各部门年龄在30岁以上的员工平均薪资，并且保留平均薪资大于10000的部门
      select post,avg(salary) from emp
      where age>30
      group by post
      having avg(salary)>10000
      ;
#distinct 去重
  ^必须是完全一样的数据才可以去重
  ^考虑主键，有主键不可能去重

    select distinct id,age from emp; （无法去重）
    select distinct age from emp;    （可以去重）

#order by 排序
  select * from emp order by salary;
    ^order by 默认升序 asc 该asc省略
    ^order by desc 降序
    ^可以跟多个
    select * from emp order by age desc,salary asc;
      ^先按第一个排，如果有相同的再按第二个排

    统计各部门年龄在30岁以上的员工平均薪资，并且保留平均薪资大于10000的部门
      select post,avg(salary) from emp
        where age>30
        group by post
        having avg(salary)>1000
        order by avg(salary) desc;

#limit  限制展示条数
  select * from emp limit 3;
    ^只拿三条数据
```

```
    select * from emp limit 0,5;
      ^从第一条数据开始拿到五条数据
      ^左开右闭
      ^起始位置，展示条数

  #正则表达式
    适用正则
       select * from emp where name regexp '^j.*(n|y)$'
```

```
##############################################################################
#表与表之间建立联系
  #表与表之间的关系  双向考虑
    一对多关系
      没有多对一一说，都是一对多关系
    多对多关系

    一对一关系

    没关系

  #外键（用于帮助我们建立表与表之间关系的特殊字段）
    foreign key
      ^一对多表关系  外键字段在多的一方
      ^在创建表的时候，一定要先建立被关联表
      ^在录入数据的时候，也必须先录入被关联表的数据
  ###一对多表关系
    #建立表关系
      create table dep(
        id int primary key auto_increment,
        dep_name varchar(16),
        dep_desc varchar(32)
        );
      create table emp(
        id int primary key auto_increment,
        name varchar(16),
        gender enum('male','female')  default 'male',
        dep_id int,
        foreign key(dep_id) references dep(id)
        );
      insert into dep(dep_name,dep_desc) values('学习部','学习很好'),('权益部','蛮好的');
      insert into emp(name,dep_id) values('gh',2);
      #此时无法直接修改dep表里的id字段
      update dep set id=200 where id=2; 不能改
      #不能直接删除dep表里的数据
      delete from t;  不能删
        删除方法：
          1.先删除学习部对应的员工数据，之后再删除部门
            操作繁琐
          2.真的做到数据同步更新，删除等
```

```
            更新一个其他跟着更新
            删除一个，都跟着删除
        create table dep(
            id int primary key auto_increment,
            dep_name varchar(16),
            dep_desc varchar(32)
            );
        create table emp(
            id int primary key auto_increment,
            name varchar(16),
            gender enum('male','female')  default 'male',
            dep_id int,
            foreign key(dep_id) references dep(id)
            on update cascade   #同步更新
            on delete cascade#同步删除
            );

###多对多表关系
  #建立表关系
    ^针对多对多关系，不能在原有两张表中创建
    需要创建一张中间表，专门存放对应关系
    create table book(
        id int primary key auto_increment,
        title varchar(32),
        price int
    );
    create table author(
        id int primary key auto_increment,
        name varchar(32)
        );
    create table con_book_auther(
        id int primary key auto_increment,
        book_id int,
        auth_id int,
        foreign key(book_id) references book(id)
        on update cascade
        on delete cascade,
        foreign key(auth_id) references author(id)
        on update cascade
        on delete cascade
        );
    insert into table ...
    ...

###一对一关系
  #举例
    id name age addr phone hobby email...
    如果一个表的字段特别多，每次查询又不是所有的字段都能用到
    将表一分为二
        用户表
            id name
        用户详情表
            id age phone...

    客户表和学生表
        报名前是客户
        报名后是学生（有客户不会报名变成学生）
```

```
    #建立表关系
      ^建在任何一方都可以，推荐建立在查询较频繁的表中
      create table authordetail(
        id int primary key auto_increment,
        phone int,
        addr varchar(32)
        );
      create table author(
        id int primary key auto_increment,
        name varchar(32),
        authordetail_id int unique,
        foreign key(authordetail_id) references (authordetail)
        on update cascade
        on delete cascade
        );
################################################################################
#多表查询
  #连表操作
          select * from dep,emp;   #结果为 笛卡尔积
          select * from emp,dep where emp.dep_id = dep.id;

    #有特定方法
      inner join 内连接
      left join   左连接
      right join 右连接
      union      全连接

    select * from emp inner join dep on emp.dep_id = dep.id;
      ^只拼接两张表中有联系的数据

    select * from emp left join dep on emp.dep_id = dep.id;
      ^以左表为主表，没有对应的就用null补充

    select * from emp right join dep on emp.dep_id = dep.id;
      ^以右表为主表，没有对应的就用null补充

    select * from emp left join dep on emp.dep_id = dep.id;
    union
    select * from emp right join dep on emp.dep_id = dep.id;
      ^左右两表所有的数据都展示，没有就null填充


  #子查询的概念
    ^将第一个查询的结果当作第二哥查询的条件用
    1.获取部门id
    2.在去员工表里筛选出对应员工
      select id from dep where name='技术' or name='人力资源';
      select name from emp where dep_id in (id号);

      合成：
      select name from emp where dep_id in (select id from dep where name='技术' or name='人力资源');

    表的查询结果可以做其他表的查询条件
    也可以通过起别名的方式，把它作为一个虚拟表和其他表相连
```

#几个关键字的执行顺序
书写顺序
select id,name from emp where id>3;
执行顺序
from---->where---->select