MySQL

```
'''
cmd下 以管理员的身份进行数据库的配置
/bin 下 --- 客户端连接服务端命令
mysql -h 127.0.0.1 -P 3306 -uroot -p

mysql 以 ； 结尾
'''
---mysql主要存储引擎---
innodb  5.5以后默认  --- 存储更加安全 表结构与表数据分开存储
myisam  5.5之前默认  --- 存储速度快   表结构 表数据 表索引
memory  内存 ---只有表结构 数据在内存
blackhole 无论存什么 都立刻消失  ---只有表结构
'''
show engines; ---查看存储引擎

创建表时可以指定引擎 engines=innodb

'''
# 环境变量配置及系统服务
tasklist
tasklist |findstr mysql 查看当前进程
taskkill /F /PID PID号
mysqld --install ---将mysql制作成系统服务
mysqld --remove  ---取消mysql系统服务
mysqladmin -uroot -p原密码 password 新密码  ---修改 设置密码

--- 破解密码
mysqld --skip-grant-tables
mysql -uroot -p

update mysql.user set password=password(123456) where user='root' and host='localhost';

flush privileges;
```

配置 ini

```
配置文件 ini
[mysqld]
[musql]
[client]
新建一个 my.ini文件
[mysql]
port=3306
basedir=C:/Program Files/MySQL/MySQL Server 5.7/
datadir=C:/ProgramData/MySQL/MySQL Server 5.7/Data


character-set-server=gb2312
```

```
default-storage-engine=INNODB
sql-mode="STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION"

max_connections=100
query_cache_size=0
table_cache=256
tmp_table_size=35M
thread_cache_size=8
myisam_max_sort_file_size=100G
myisam_sort_buffer_size=69M
key_buffer_size=55M
read_buffer_size=64K
read_rnd_buffer_size=256K
sort_buffer_size=256K
```

命令

```
show databases; --- 查看数据库
quit
exit --- 退出数据库
\c 取消错误命令
\s 查看当前信息

create database db1;  ---增库
create database db2 charser='gbk';   ---增库

show databases; ---查库
show create database db;  ---查单个库

alter database db charset='utf8'; ---改

drop database db; ---删库跑路

---表 的增删改查---
select database(); ---查看当前库名
use db; --- 使用某库

create table  tab(id int, name char);  --- 增加表
show  tables;   --- 查看当前库中的表
desc tab; ---查看表结构
alter table tab modfiy name char(16);  ---改表
drop table tab; ---删表

---数据 的增删改查---
insert into tab values();
insert into tab values(),(),();  --- 增

select * from tab;
select 字段 from tab;  ---查

update tab set 字段='' where 条件;  ---修改

delete from tab where 条件;  ---删
```

```
create table 表名(字段名1 类型(宽度) 约束条件，字段名1 类型(宽度) 约束条件，字段名1 类型(宽度) 约束条件); --- 创建表的完整语法
同一张表中字段名不可重复  宽度和约束条件可选 字段和类型必须

---数据类型---
--整型--
TINYINT
SMALLINT
MEDUIMINT
INT （括号内的数字不是限制位数  而是显示长度）
BIGINT  ------整型数据不需要添加宽度

--浮点型--
FLOAT(255,30)
DOUBLE(255,30)
DECIMAL(65,30)

--字符类型--
char  -- 定长 （超出报错 不足用空格补全）
varchar  -- 变长（超出报错 不足用空格补全有几个存几个）

--时间类型--
date  ---年月日
datetime  ---年月日时分秒
time  ---时分秒
year  ---年

--枚举与集合--
enum  ---多选一
eunm('male', 'female', 'others')
set  ---多选多
set('read', 'female', 'others')

---约束条件---
null 可以为空
not null 不可为空
unsigned 无符号
default 默认值
unique 唯一
--- 单例唯一
    直接在字段后添加 unique
--- 联合唯一
    单个可重复 组合不可重复
    unique(字段1, 字段2)
primary key 主键 （非空且唯一）
auto_increment 自增
```

表与表之间的关系

```
---外键---

建立表与表之间特殊关系的字段 foreign key
```

```
如何考虑表关系：分别从俩张表的角度出发
建表时需要先建被关联表
---一对多---
    外键字段建在多的一张表中
    foreign key(字段名) references 表名(id)
    on update cascade  --- 级联更新（同步跟新）
    on delete cascade  --- 级联删除（同步删除）
---多地多---
    使用第三张表来建立多对多的关系
    foreign key(字段名) references 表名(id)
    on update cascade  --- 级联更新（同步跟新）
    on delete cascade  --- 级联删除（同步删除）
--实例
create table book2author(
    id int primary key auto_increment,
    book_id int,
    author_id int,
    foreign key(author_id) references author(id)
    on update cascade  --- 级联更新（同步跟新）
    on delete cascade,  --- 级联删除（同步删除）
    foreign key(book_id) references book(id)
    on update cascade  --- 级联更新（同步跟新）
    on delete cascade  --- 级联删除（同步删除）
)
---一对一---
    外键建立在查询较多的表中
    外键字段需要添加唯一约束
    foreign key(本表字段) references 被关联表名(id)
    on update cascade  --- 级联更新（同步跟新）
    on delete cascade  --- 级联删除（同步删除）
```

修改表和复制表

```
---修改表---
alter table 表名 rename 新表名;  ---修改表名
alter table 表名 add 字段名 字段类型(宽度) 约束条件;  ---添加字段
alter table 表名 add 字段名 字段类型(宽度) 约束条件 first;  ---添加字段
alter table 表名 add 字段名 字段类型(宽度) 约束条件 after 字段名;  ---添加字段
alter table table 表名 drop 字段名;  ---删除字段
alter table 表名 modify 字段名 字段类型(宽度) 约束条件;  ---修改字段
alter table 表名 change 旧字段名 新字段名 字段类型(宽度) 约束条件;  ---修改字段
---复制表---
create table 表名 select * from 旧表;  不能赋值主键 外键
```

表查询

```
---关键字的执行顺序---
"""
select * 占位  补全后续语句 最后将*替换为字段
"""
聚合函数 :  max(字段)  min(字段)  avg(字段)  sum(字段)  count(字段)
```

```
as 可以给字段起别名 也可以为表起别名
where  ---约束条件  用于筛选
    字段 between 左边界 and 右边界  等价于  字段 >= 左边界 and 字段 <= 右边界
    字段=值1 or 字段=值2 or 字段=值3 or 等价于 字段 in (值1, 值2, 值3)
    like 模糊查询
        % 匹配任意多个字
         _ 匹配任意单个字符
     is null   is not null

group by ---分组
    group by 字段; 非严格模式下 可以执行 返回每个组的第一条数据 严格模式下报错
    设置严格模式之后 分组默认只能拿到分组的依据
    set global sql_mode = 'strict_trans_tables, only_full_group_by';
    聚合函数 :  max(字段)  min(字段)  avg(字段)  sum(字段)  count(字段)
                count的字段不能为null
    group_concat() ---支持获取分组后的其他字段 , 且可凭借
    concat() ---不分组时使用
    ----注意事项
    where 与 group by 同时出现 where必须在前
    聚合函数只能在分组之后使用  默认整张表就是一组

   having ---分组之后的筛选条件
   distinct ---去重  必须是完全一样的数据才可以去重
   select distinct 字段1, 字段2 from ...

   order by 字段1, 字段2  默认升序       asc 升序  desc降序 --跟在字段后
   limit ---限制展示条数
   limit 展示条数
   limit 起始位置, 展示条数
正则相关: regexp
```

多表查询

```
inner join  ---内连接
只拼接两表共有部分
select * from emp inner join dep on emp.dep_id =dep.id;

left join  ---左连接
左表全显示 没有对应的就显null
select * from emp left join dep on emp.dep_id =dep.id;

right join  ---右连接
右表全显示 没有对应的就显null
select * from emp right join dep on emp.dep_id =dep.id;

union  ---全连接
左右俩表全部显示
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;


子查询 ---将一个查询语句的结果当作另外一个查询语句的条件去用
```

查看严格模式

```
show variables like "%mode"

% 匹配任意多个字
_ 匹配任意单个字符

---修改严格模式---
set session   ---当前窗口有效
set global  ---全局有效
set glbal sql_mode = 'STRICT_TRANS_TABLES';

set glbal sql_mode = 'STRICT_TRANS_TABLES, PAD_CHAR_TO_FULL_LENGTH';
select char_length(字段) from tab; ---查询表中字段长度

truncate tab;  清空主键并重置表
delete from 删除表中数据 主键的自增不会停止
```

pymysql

```
pip install pymysql  # 安装
import pymysql
conn = pymysql.connect(
    'host' = '',
    port = '',
    user = '',
    password = '',
    charset = 'utf8'
    database = ''
)
cursor = conn.cursor()
cursor = conn.cursor(cursor=pymysql.cursor.Dictcursor) # 以字典形式返回
sql = 'sql语句'
res = cursor.execute(sql)
cursor.fetchone()# 获取查询结果
cursor.fetchall()# 获取查询结果
cursor.fetchmany(2)# 获取查询结果 指定查询数目
cursor.scroll()  # 相对于光标后移relative  相对于起始位后移absolute

'''sql注入'''
利用语法特性 书写一些特定3的语句实现查询

# 敏感的数据不要自己做拼接，直接交给execute操作
res = cursor.execute(sql, (username, password))
```

视图（了解 不推荐使用）

```
--- 试图就是通过查询一张虚拟表保存下来 下次可以直接使用
--创建视图
select * from tab1 inner join tab2 on tab1.id = tab2.tab1_id;
create view 表名 as 虚拟表查询sql语句

--- 创建视图在银盘上只会有表结构 没有表数据 修改里边的数据可能会影响原表中的数据
```

触发器

```
---在满足对数据表进行增 删 改 自动触发 （实现监控 日志 。。。）
create trigger 触发器名字 before/after insert/update/delete on 表名
for each row
begin
    sql语句
end

---名字见名知意
tri_before_insert_tab

delimiter $$ --修改sql语句默认结束符  只作用于当前窗口

----------------------
if 条件 then
    ...
end if
---删除触发器
drop 触发器名字
```

事务 --- ACID --原子性 一致性 隔离性 持久性

```
--- 开启一个事务 可以包含多条sql语句  同时成功 同时失败 ---事务的原子性
---使用事务
start transaction;  ---开启事务
rollback;  ---回滚
commit;  ---二次确认 （确认后无法回滚）
```

存储过程

```
--- 类似于python中的自定义函数 将一系列sql语句存储在mysql服务端 使用时直接调用
create procedure 存储过程名字()
begin
    sql语句
end
call 存储过程名字():
```

函数

```
NOW()  ---显示当前时间
date_format()  ---定制化显示数据
```

流程控制

```
---if 判断
if 条件 then
    ...
else if条件 then
    ...
else ...
end if;
---while 循环
while 条件 do
...
end while;
```

索引

```
--- 数据都时存储在硬盘上的 查询数据不可避免需要进行io操作
索引 就是一种数据结构 类似于数目目录 提升查询速度 降低io操作
索引在mysql中也叫做 键

primary key --- 聚集索引
unique key --- 辅助索引
以上俩种有约束 以下一种无约束
index key --- 辅助索引

---缺点:
1 大量数据存在 创建索引熟读会 很慢
2 一旦数据被修改 需要重新创建索引
```

B+树

```
--- 只有叶子节点存放真实数据 其他节点都是存虚拟数据 仅仅用于指路
--- 树的层级越高查询经历的操作越多
```