**综合使用 查询 目录：**

#----综合使用

书写顺序

select distinct \* from '表名' where '限制条件' group by '分组依据' having '过滤条件' order by limit '展示条数'

执行顺序

from -- 查询

where -- 限制条件

group by -- 分组

having -- 过滤条件

order by -- 排序

limit -- 展示条数

distinct -- 去重

select -- 查询的结果

正则：select \* from emp where name regexp '^j.\*(n|y)$';

集合查询：max 、min 、avg 、sum 、count 、group\_concat 。

内连接：inner join

左连接：left join

右连接：right join

全连接： 左连接 union 右连接

replace 替换

拼接：concat、concat\_ws、group\_concat

**常规设置操作**

**1.服务器设置远程访问**

grant all privileges on \*.\* to 'root'@'%' identified by '123456' with grant option;

**2.Linux中数据库的基本操作命令**

1.使用service

启动：service mysql start

停止：service mysql stop

重启：service mysql restart

2.清屏：clear,reset

**3.备份数据库**

#mysqldump -uroot -p密码 数据库名 > D:/备份文件名.sql

**4.恢复备份的数据库**

#首先在mysql里建好数据库名

#mysql -uroot -p密码 数据库名 < D:/备份文件名.sql

**5.查询binlog日志是否开启**

show variables like 'log\_%';

**基本操作：**

**1.单表约束**

#主键约束：PRIMARY KEY 要求被装饰的字段：唯一和非空

#唯一约束：UNIQUE 要求被装饰的字段：唯一，

# .联合唯一：在结尾：unique(字段1，字段2)

#非空约束：NOT NULL 要求被装饰的字段：非空

#外键约束：FOREIGN KEY 某主表的外键

#自动增加：AUTO\_INCREMENT 自动增加(需要和主键 PRIMARY KEY 同时用)

#设置默认值:DEFAULT 为该属性设置默认值

# 在int、char中：zerofill 不足位数默认填充0

#

#

**2.常用数据类型**

int #整型，4个字节 一般不需要指定宽度，（8）：只是显示为8位，默认有负号设置无负号： unsigned

double #浮点型，例如double(5,2),标识最多5位，其中2位为小数，即最大为999.99

varchar #可变长度字符串类型。例如：varchar(10) 'aaa' 占3位

char #固定长度字符串类型。例如：char(10) 'aaa' 占10位

text #大文本字符串类型。

blob #字节类型。例如：

datetime #日期时间类型。例如：datetime(yyyy-MM-dd hh:mm:ss)

date #日期类型。例如：date(yyyy:MM:dd)

time #时间类型。例如：time(hh:mm:ss)

timestamp #时间戳类型。例如：timestamp(yyyy-MM-dd hh:mm:ss) 会自动赋值

enum #枚举 多选一 enum('male','female')，default为默认值

例如：sex enum('male','female') not null default 'male'

set #集合 多选多,可以选一个 set（'read','DJ','DBJ','run')

注：字符串类型和时间类型都要用单引号括起来，空值为null

**3.查看数据列表**

show databases; -- 查看所有数据库

show create table 表名; -- 查看表的创建细节

desc 表名; -- 查看表结构

**4.进入数据库**

#use 数据名

use Python\_7

**5.创建数据库**

#CREATE DATABASE 数据库名

CREATE DATABASE Python\_7;

CREATE DATABASE pyrhon\_7 charset utf8

# 修改数据库编码

alter database db1 charset gbk;

**6.删除数据库**

#drop database 需要删除的数据库名

drop database Python\_7;

**7.查看表**

select database(); # 查看当前所在的库

show tables; -- 查看数据库中所有表

desc 表名; -- 查看表结构

show create table 表名; -- 查看表的创建细节

**8.创建表**

# 创建新表

# create table 新建数据表名（

# 字段名 类型(长度) [约束(具体见1)],

# 字段名 类型(长度) [约束(具体见1)]

# );

create table class(

id INT AUTO\_INCREMENT,

name varchar(32) UNIQUE,

age varchar(32) NOT NULL

);

#需要注意

#根据已有的表创建新表

create table 新表 like 旧表 -- 使用旧表创建新表

create table 新表 as select 字段1 字段2... from definition only -- 使用自定义值去新建表

**9.删除表**

#drop table 数据库表名

drop table Python

**10.修改表**

alter table 表名 add 字段名 类型（长度） [约束]; -- 添加列

alter table 表名 modify 字段名 类型（长度） [约束]; -- 修改列的类型长度及约束

alter table 表名 change 旧字段名 新字段名 类型（长度） [约束]; -- 修改列表名

alter table 表名 drop 字段名; -- 删除列

alter table 表名 character set 字符集; -- 修改表的字符集

rename table 表名 to 新表名; -- 修改表名

**11.增加数据**

insert into 表(字段名1，字段名2..) values(值1，值2..);-- 向表中插入某些列

insert into 表 values(值1，值2，值3..); -- 向表中插入所有列

**12.修改数据**

update 表名 set 字段名=值,字段名=值...; -- 这个会修改所有的数据，把一列的值都变了

update 表名 set 字段名=值,字段名=值... where 条件; -- 只改符合where条件的行

**13.删除数据**

delete from 表名 -- 删除表中所有记录

delete from 表名 where 条件 -- 删除符合 where条件的数据

truncate table 表名; -- 把表直接drop掉，重新建表，auto\_increment将置为零。删除的数据不能找回。执行速度比delete快

**14.数据的简单查询**

select \* from 表名; -- 查询所有列

select 字段名1,字段名2,字段名3.. from 表名; -- 查询指定列

**15.几个简单的基本的sql语句**

select \* from 表名 where 范围 -- 选择查询

insert into 表名(field1,field2) values(value1,value2) -- 插入

delete from 表名 where 范围 -- 删除

update 表名 set field1=value1 where 范围 -- 更新

select \* from 表名 where field1 like ’%value1%’ -- 查找

select \* from 表名 order by field1,field2 [desc] -- 排序：

select count as 需要统计总数的字段名 from 表名 -- 总数

select sum(field1) as sumvalue from 表名 -- 求和

select avg(field1) as avgvalue from 表名 -- 平均

select max(field1) as maxvalue from 表名 -- 最大

select min(field1) as minvalue from 表名 -- 最小

**16.存储引擎**

# 查看所有的存储引擎

show engines;

# 查看不同存储引擎存储表结构文件特点

create table t1(id int)engine=innodb; -- MySQL默认的存储引擎,支持事务,支持行锁，支持外键。有且只有一个主键，用来组织数据的依据

create table t2(id int)engine=myisam; -- 不支持事务，不支持外键，支持全文索引，处理速度快。

create table t3(id int)engine=blackhole; -- 黑洞，写入它的任何内容都会消失

create table t4(id int)engine=memory;-- 将表中的数据存储在内存中。表结构以文件存储于磁盘。

insert into t1 values(1);

insert into t2 values(1);

insert into t3 values(1);

insert into t4 values(1);

**17.设置严格模式**

# 查询

show variables like '%mode%';

# 设置

set session -- 设置当前窗口下有效

set global -- 全局有效，终身受用

set global sql\_mode = "STRICT\_TRANS\_TABLES";

# 设置完成后需要退出客户端，重新登录客户端即可，不需要重启服务端

group by分组涉及到的模式：

设置sql\_mode为only\_full\_group\_by，意味着以后但凡分组，只能取到分组的依据，

不应该在去取组里面的单个元素的值，那样的话分组就没有意义了，因为不分组就是对单个元素信息的随意获取

"""

set global sql\_mode="strict\_trans\_tables,only\_full\_group\_by";

# 重新链接客户端

**18.like 的用法**

A:% 包含零个或多个字符的任意字符串：

1、like'Mc%' 将搜索以字母 Mc 开头的所有字符串（如 McBadden）。

2、like'%inger' 将搜索以字母 inger 结尾的所有字符串（如 Ringer、Stringer）。

3、like'%en%' 将搜索在任何位置包含字母 en 的所有字符串（如 Bennet、Green、McBadden）。

B:\_（下划线） 任何单个字符：

like'\_heryl' 将搜索以字母 heryl 结尾的所有六个字母的名称（如 Cheryl、Sheryl）。

C：[ ] 指定范围 ([a-f]) 或集合 ([abcdef]) 中的任何单个字符：

1，like'[CK]ars[eo]n' 将搜索下列字符串：Carsen、Karsen、Carson 和 Karson（如 Carson）。

2、like'[M-Z]inger' 将搜索以字符串 inger 结尾、以从 M 到 Z 的任何单个字母开头的所有名称（如 Ringer）。

D：[^] 不属于指定范围 ([a-f]) 或集合 ([abcdef]) 的任何单个字符：

like'M[^c]%' 将搜索以字母 M 开头，并且第二个字母不是 c 的所有名称（如MacFeather）。

E：\* 它同于DOS命令中的通配符，代表多个字符：

c\*c代表cc,cBc,cbc,cabdfec等多个字符。

F：？同于DOS命令中的？通配符，代表单个字符 :

b?b代表brb,bFb等

G：# 大致同上，不同的是代只能代表单个数字。k#k代表k1k,k8k,k0k 。

下面我们来举例说明一下：

例1，查询name字段中包含有“明”字的。

select \* from table1 where name like '%明%'

例2，查询name字段中以“李”字开头。

select \* from table1 where name like '李\*'

例3，查询name字段中含有数字的。

select \* from table1 where name like '%[0-9]%'

例4，查询name字段中含有小写字母的。

select \* from table1 where name like '%[a-z]%'

例5，查询name字段中不含有数字的。

select \* from table1 where name like '%[!0-9]%'

以上例子能列出什么值来显而易见。但在这里，我们着重要说明的是通配符“\*”与“%”的区别。

很多朋友会问，为什么我在以上查询时有个别的表示所有字符的时候用"%"而不用“\*”？先看看下面的例子能分别出现什么结果：

select \* from table1 where name like '\*明\*'

select \* from table1 where name like '%明%'

大家会看到，前一条语句列出来的是所有的记录，而后一条记录列出来的是name字段中含有“明”的记录，所以说，当我们作字符型字段包含一个子串的查询时最好采用“%”而不用“\*”,用“\*”的时候只在开头或者只在结尾时，而不能两端全由“\*”代替任意字符的情况下。

**高级查询操作**

**1、外键表创建**

一对多（Foreign Key）

# foreign key(需要关联的本字段) references 需要关联对表的表(需要关联对表的字段)

例如：

创建dep

foreign key(dep\_id) references dep(id)

# 同步更新，同步删除

on update cascade #同步更新

on delete cascade #同步删除

**2、表复制**

复制表

create table t1 select \* from test;

只复制表结构

create table t1 select \* from test where 1=2;

**3、单表查询查询**

**0.综合使用**

#----综合使用

书写顺序

select distinct \* from '表名' where '限制条件' group by '分组依据' having '过滤条件' order by limit '展示条数'

执行顺序

from -- 查询

where -- 限制条件

group by -- 分组

having -- 过滤条件

order by -- 排序

limit -- 展示条数

distinct -- 去重

select -- 查询的结果

正则：select \* from emp where name regexp '^j.\*(n|y)$';

集合查询：max 、min 、avg 、sum 、count 、group\_concat 。

拼接：concat、concat\_ws、group\_concat

内连接：inner join

左连接：left join

右连接：right join

全连接： 左连接 union 右连接

**1.where 查询**

# between 在...之间

select id,name from emp where id >= 3 and id <= 6;

相当于：

select \* from emp where id between 3 and 6;

# or 或者

select \* from emp where id >= 3 or id <= 6;

# in，后面可以跟多个值

select \* from 表名 where 字段名 in (条件1,条件2,条件三);

# like （见上18）

# char——length() 取字符长度

select \* from 表名 where char\_length(需要获取长度的字段名) = 4;

not 配合使用

注意：判断空不能用 = ，只能用 is

**2.group by 分组**

select 查询字段1,查询字段2,... from 表名

where 过滤条件

group by分组依据 # 分组后取出的是每个组的第一条数据

**3.聚合查询 ：以组为单位统计组内数据>>>聚合查询(聚集到一起合成为一个结果)**

# max 最大值

# 每个部门的最高工资

select post,max(salary) from emp group by post;

# min 最小值

# 每个部门的最低工资

select post,min(salary) from emp group by post;

# avg 平均值

# 每个部门的平均工资

select post,avg(salary) from emp group by post;

# 每个部门的工资总和

# sum 求和

select post,sum(salary) from emp group by post;

# count（需要计数字段） 计数

# 每个部门的人数

select post,count(id) from emp group by post;

# group\_concat(需要分组后的字段) # 不仅可以用来显示除分组外字段还有拼接字符串的作用

select post,group\_concat(name) from emp group by post;

-- post：分组字段，name 需要分组后显示的字段

拼接：

concat（不分组时用）拼接字符串达到更好的显示效果 as语法使用

举例：

select concat("NAME: ",name) as 姓名 from emp;

concat\_ws: 如果拼接的符号是统一的可以用

举例：

select concat\_ws(':',name,age,sex) as info from emp;

group\_concat：

举例：

select post,group\_concat(name,'DSB') from emp group by post;

补充：as语法 起别名

select name as 姓名,salary as 薪资 from emp;

**4.having 过滤查询**

# having的语法格式与where一致，只不过having是在分组之后进行的过滤，即where虽然不能用聚合函数，但是having可以！

# 用法

select 查询字段1,查询字段2,... from 表名

where 过滤条件1

group by分组依据

having avg(过滤条件2) > 10000;

**5.distinct 去重**

# 对有重复的展示数据进行去重操作

select distinct 需取重字段 from 表名;

**6.order by 排序**

select \* from emp order by salary asc; #默认升序排

select \* from emp order by salary desc; #降序排

# 多条件排序

#先按照age降序排，在年轻相同的情况下再按照薪资升序排

select \* from emp order by age desc,salary asc;

**7.limit 限制展示条数**

# 限制展示条数

select \* from emp limit 3;

# 查询工资最高的人的详细信息

select \* from emp order by salary desc limit 1;

# 分页显示

select \* from emp limit 0,5; # 第一个参数表示起始位置，第二个参数表示的是条数，不是索引位置

select \* from emp limit 5,5;

**8.正则**

select \* from emp where name regexp '^j.\*(n|y)$';

**9.replace 替换**

replace(str1,old,new) -- str1：需要替换的字段名

update gd\_km set mc=replace（mc,'土地','房子'）

说明：new替换str1中出现的所有old，返回新的字符串，如果有某个参数为NULL，此函数返回NULL

该函数可以多次替换，只要str1中还有old存在，最后都被替换成new

若new为空，则删除old

**四、多表查询**

**1.内连接、左连接、右连接、全连接**

1、内连接：只取两张表有对应关系的记录（只拼两个表共有的）

左表 inner join 右表 on 条件

select \* from emp inner join dep on emp.dep\_id = dep.id

where dep.name = "技术";

2、左连接：在内连接的基础上，保留左边的数据，右边没有就为空

左表 inner left 右表 on 条件

3、右连接：在内连接的基础上，保留右边的数据，左边没有就为空

左表 inner right 右表 on 条件

4、全连接：左右连接都有，用union连接

左表 inner left 右表 on 条件 union 左表 inner right 右表 on 条件

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;

**2.子查询**

# 就是将一个查询语句的结果用括号括起来当作另外一个查询语句的条件去用

select name from where id in(select dep\_id from emp where age>25);