高级查询

2024年6月25日 15:25

四、高级查询

4.1 查询处理

4.1.1 排序 order by

默认不是按照主键进行排序的

order by 子句:对查询结果按指定字段进行排序。也可以指定 select列表中列的序号进行排序

```
# 按照empno升序查询(asc),默认升序
select * from emp order by empno asc;
# 降序查询
select * from order by empno desc;

create table if not exists users
(
  id int primary key,
  name varchar(20)
);

insert into users values(1,"zhangsan"),(2,"lisi"),(3,"wangwu");
select * from users; # 默认不是按照主键排序的
alter table users add age int; # 添加一个字段
select * from users; # 排序方式会改变

# 按照多个字段排序
select * from emp order by deptno,sal desc; # 按照deptno升序，deptno相同的话，按照sal降序
```

4.1.2 限制数量 limit

在真实的业务中，数据的条数是很大的，对数据集进行排序后，不可能之间显示全部的数据，一般只是输出几条数据。这时就就要用到 limit 来限制显示数据条数

limit 子句:select 语句返回所有匹配的行，它们可能是指定表中的每个行。为了前几行或中间几行，可使用

limit 子句。使用limit 可以解决分页问题。

```
limit 行数 (从第一行开始)
limit  开始行 (从 0 开始) , 行数
```

```
# 显示几条数据
select * from emp order by empno asc limit 5;
# 从第3+1条开始，显示5条数据
select * from emp order by empno asc limit 3,5;
```

4.1.3 去重 distinct

distinct 关键字:用于返回唯一不同的值

```
# 查询 有几种工作
select distinct job from emp;
# 同时作用两列，不能查询目标列以外的列
select distinct job,mgr from emp; # 两个字段不能同时重复
```

4.1.4 组合查询 union

union 操作符:执行多个查询(多条 select语句)，将结果合并为单个结果集返回。

注意：查询时，字段数量一致，字段类型相似
union 会自动去重，不想去重 用 union all

```
select 字段1[,字段2,...] from 表1
union
select 字段1[,字段2,...] from 表2;
```

```
# 组合查询，按照行排列
select empno,ename from emp
union
select deptno,dname from dept;
```

4.2 函数

4.2.1 数值函数

```
# abs(x):返回x的绝对值
select abs(-20); # 20

# ceil(x):向上取整，返回大于等于x的最小整数值
select ceil(5.3); # 6

# floor(x):向下取整，返回小于等于x的最大整数值
select floor(5.99); # 5

# round(x,y=0):四舍五入，将x四舍五入y位小数，y不传返回整数，y为负数时，保留 x值到小数点左边 y位
select round(5.5);

# truncate(x,y):截断函数，返回被舍去至小数点后y位的数字x，y为负数时截断小数点左边y位
select truncate(5.23,1);

# mod(x,y):返回x除以y的余数
select mod(5,2);
```

```
# rand():生成 0-1 的随机数
select rand();

# rand():生成 1-10 的随机数
select truncate(rand()*10+1,0);
```

4.2.2 字符函数

(1) concat(s1,s2,...):字符串连接，如果任何一个参数为 nul，则返回值为 null
```
select concat("hello ","world");
```

(2) concat_ws(x,s1,s2,….):指定分隔符的字符连接函数，x是连接分隔符，如果分隔符为null，则结果为 null。
```
select concat_ws("xxx","hello ","world");
```

(3) lower(str):大写转小写
```
select lower("WeiHong");
```

(4) upper(str):小写转大写
```
select upper("weihong");
```
(5) length(str):字符串长度
(6) ltrim(str):删除字符串左侧空格
(7) rtrim(str):删除字符串右侧空格
(8) trim(str):删除字符串两侧空格
(9) substr(str,n,len):截取子字符串，字符串 str 从n的位置截取长度为 len 的字符串，如果 n为负数，则子字符串的位置起始于字符串结尾的n个字符
(10) left(str, n):返回字符串 str 的最左边n个字符
(11) right(str,n): 返回字符串 str 的最右边 n个字符
(12) replace(str,from str,to str):替换函数，字符串 str 中所有的字符串 from str均被 to str 替换，然后返回这个字符串
(13) format (x,n):将数字x格式化，并以四舍五入的方式保留小数点后n位，结果以字符串的形式返回。若n为0，则返回结果不含小数部分。

4.2.3 日期时间函数

(1) curdate()/current_date():获取当前日期，YYYY-MM-DD 格式
(2) curtime()/current_time():获取当前时间，HH:MM:SS 格式
(3) week(date):返回 date 为一年中的第几周
(4) now()/sysdate():获取当前日期和时间，YYYY-MM-DD HH:MM:SS 格式
(5) date_add(date,interval expr type):执行日期的加运算，date 是一个 datetime 或 date 值，指定起始时间。expr是时间间隔。type为关键词，如YEAR,MONTH,DAY,WEEK HOUR等。
(6) datediff(date1,date2):计算两个日期之间的间隔天数
(7) unix_timestamp(date):返回 date 的 UNIX 时间戳
(8) form_unixtime(unix):返回 unix 时间戳的日期值
(9) date_format(date,format):日期格式化，按 format 格式化 date 值
(10) str_to_date(date,format):将字符串转换成 date 类型

```
# (9) date_format(date,format):日期格式化，按 format 格式化 date 值
select date_format('2024/6/25','%Y年%m月%d日');

# (10) str_to_date(date,format):将字符串转换成 date 类型
select str_to_date('2024年6月+++++++++25日','%Y年%m月+++++++++%d日');
```
将 2024年6月+++++++++25日 转为 2024-06-25

4.2.4 聚合函数(分组函数)

聚合函数会改变数据的条数,查询后变为一条数据

avg(expression):返回某列的平均值
sum(expression):返回某列值的和
count(expression):返回某列的行数
max(expression):返回某列的最大值
min(expression):返回某列的最小值

注意:

聚合函数会自动的忽略空值，不需要手动增加条件排除 NULL

聚合函数不能作为 where 子句后的限制条件（因为在查的过程，聚合函数无法计算，得全部查完才能算）

```
# avg(expression):返回某列的平均值
select avg(sal) from emp;
# sum(expression):返回某列值的和
select sum(sal) from emp;
# count(expression):返回某列的行数
select count(sal) from emp;
# max(expression):返回某列的最大值
select max(sal) from emp;
# min(expression):返回某列的最小值
select min(sal) from emp;
```

4.2.5 流程函数

if(value,t,f):如果 value 为真返回 t，否则返回 f

ifnull(column, value):如果 column 为空返回 value，否则返回 column

在 SQL 语句当中若有 NULL 值参与数学运算，计算结果一定是 NULL，为了防止计算结果出现 NULL，建议先使用 ifnu 空值处理函数预先处理。

```
# 任何数 + null = null
select sal + comm from emp; # 这样的结果大多都是null

select sal + ifnull(comm,0) from emp;

select ename,empno,if(sal >3000,"sal_high","sal_low") from emp;
```

4.2 分组查询 group by

4.2.1 创建分组

group by 子句:根据一个或多个字段对结果集进行分组，在分组的字段上可以使用count、sum、avg等函数。

```
select 字段1[字段2, function(字段1), function(字段2)...]
from 表
group by 字段1;
```

按照分组的字段进行分组，分组之后，一组只有一条数据，因此，分组之后，只有分组的字段是有意义的。

```
# 查看公司有几个部门
select deptno from emp group by deptno;

# 添加部门编号 为 20 的员工人数
select count(*) from emp where deptno = 20;

# 查看每个部门的人数
select deptno,count(*) from emp group by deptno;
```

注意

如果分组列中具有 NULL 值，则 NULL 将作为一个分组返回。
如果列中有多行 NULL值，它们将分为一组。
group by 子句必须出现在 where 子句之后，order by 子句之前

因此使用where 只能先过滤，再分组

4.2.2 过滤分组

having 子句:having 非常类似于 where。唯一的差别是 where 过滤行，而 having 过滤分组。
having 必须和 group by 一起使用。

having 和 where 的区别也可以理解为，**where 是分组前过滤，having 是分组后过滤**。

```
# 查看每个部门的人数,并保留部门人数大于5的分组
select deptno,count(*) from emp group by deptno having count(*)>5;

# 查看每个部门工资大于1000的人数,并保留部门人数大于2的分组
select deptno,count(*) from emp where sal > 1000 group by deptno having count(*)>2;
```

 # 过滤分组查询练习

```
# 查询出该公司有哪几种岗位及每个岗位的人数
select job,count(job) from emp group by job;


# 计算每个工作岗位的最高薪水，并且由低到高进行排序
select job,max(sal) as max_sal from emp group by job order by max_sal;

# 计算每个部门的平均薪水
select deptno,avg(sal) from emp group by deptno having avg(sal);

# 计算出不同部门不同岗位的最高薪水
select deptno,sal from emp  group by deptno having max(sal);

# 找出每个工作岗位的最高薪水，除manager 之外
select job,sal from emp where job != "manager" group by job having max(sal);

# 找出每个工作岗位的平均薪水，显示平均薪水大于2000的
select job,avg(sal) from emp group by job having avg(saL) > 2000;
```

4.3 select执行顺序

5. select 字段名	对当前临时表进行整列读取
1. from 表名	将硬盘上的表文件加载到内存
2. where ...	将符合条件的数据行摘取生成一张新的临时表
3. group by ...	根据列中的数据种类，将当前临时表划分成若干个新的临时表
4. having ...	可以过滤掉 group by生成的不符合条件的临时表
6. order by ...	对 select 生成的临时表重新排序，生成新的临时表
7. limit ...	对最终生成的临时表数据行进行截取

以上关键字的顺序不能改变，严格遵守

4.4 正则表达式

regexp 操作符，regexp 操作符后面跟的就是正则表达式，正则表达式的作用是匹配文本，将一个模式(正则表达式)与一个文本串进行比较。

like 和 regexp 的区别

like 匹配整个列，如果被匹配的文本仅在列值中出现(没有配合其他通配符)，like将不会找到它。regexp 在列值内进行匹配，如果被匹配的文本在列值中出现，regexp将会找到它，相应的行将被返回。

4.4.1 匹配单个实例

(1)|:表示匹配其中之一,使用|从功能上类似 or
(2)[]:匹配字符之一，[]是另一种形式的 or 语句。[123]为 [1|2|3]的缩写。
(3)[-]:匹配范围，使用-来定义一个范围。例如:[1-3]、[a-z]。
(4)\\ :转义字符，多数正则表达式使用单个反斜杠作为转义字符，但 MySQL 要求两个反斜杠(MySQL 自己解释一 个，正则表达式库解释另一个)。
(5)匹配字符类:存在找出你自己经常使用的数字、所有字母字符或所有数字字母字符等的匹配。为更方便工作，可以使用预定义的字符集，称为字符类。

类	说明
[[:alnum:]]	任意字母和数字（同 [a-zA-Z0-9]）
[[:alpha:]]	任意字母（同 [a-zA-Z]）
[[:blank:]]	空格和制表（同 [\\t]）
[[:cntrl:]]	ASCII 控制字符（ASCII 0 到 31 和 127）
[[:digit:]]	任意数字（同 [0-9]）
[[:graph:]]	与 [:print:] 相同，但不包括空格
[[:lower:]]	任意小写字母（同 [a-z]）
[[:print:]]	任意可打印字符
[[:punct:]]	既不在 [[:alnum:]] 又不在 [[:cntrl:]] 中的任意字符
[[:space:]]	包括空格在内的任意空白字符（同 [\\f\\n\\r\\t\\v]）
[[:upper:]]	任意大写字母（同 [A-Z]）
[[:xdigit:]]	任意十六进制数字（同 [a-fA-F0-9]）

4.4.2 匹配多个实例

- 常用元字符

元字符	说明
.	匹配任意字符
^	匹配字符串的开始，^ 在 [] 中表示否定
$	匹配字符串的结束

- 重复元字符（修饰前一个字符）

元字符	说明
*	任意个匹配
+	一个或多个匹配（等于{1,}）
?	0 个或 1 个（等于{0,1}）
{n}	指定数目的匹配
{n,}	不少于指定数目的匹配
{n,m}	匹配数目的范围（m 不超过 255）