SQL语句

SQL简单检索语句如下：

```
select prod_id,prod_name
from products
where ven_id= 1003 AND prod_price <=10
order by prod_price
limit 5;
```

关系操作符

SQL中的关系操作符，有<,>,=,!=,<>,等等。

OR,AND,IN,NOT操作符

SQL中逻辑关系操作符，AND,OR等。其中AND的优先级更高。实际中，应该加上()来表明优先级。

IN操作符，指明范围：

```
select prod_name,prod_price
from products
where ven_id IN(1000,1001,10002)
ORDER BY prod_name;
```

IN就是表示一个范围，和python中类似

```
for iter in range(10)
```

IN操作符有如下有点

- 语法简洁直观
- IN操作符一般比OR更快
- IN可以包含其他select的语句，可以动态简历where子句

Not操作符否定用

```
select prod_name,prod_price
from products
where ven_id NOT IN(1000,1001,10002)
ORDER BY prod_name;
```

通配符 (wildcard)

LIKE操作符

LIKE操作符指明接下来跟的**搜索模式**利用通配符匹配而不是直接相等匹配相比较。

谓词：LIKE是谓词而不是操作符。(略过)

%通配符

和regex中的.*作用相同。表示任意字符出现任意次数。语法如下：

```
select prod_id,prod_name
from products
where prod_name Like 'jet%';
```

其中`jet%`表示，以jet开头。如果要求jet出现在任何位置，写法如下

```
where prod_name like '%jet%';
```

> 注意NULL，NULL值会被忽略匹配。除非使用 is null 判断。

正则表达式REGEX

Mysql中的正则表达式是传统的regex的一个小的子集。但又些许差别。比如模式

```
regex r("1000|2000");// 匹配1000000或1002000
// 而Mysql中的REGEXP效果类似以下用法
regex r("(1000)|(2000))");
```

Mysql中的REGEXP的模式区分大小写。

REGEXP用法如下，

```
select prod_name
from products
where prod_name REGEXP '1000'
ORDER BY prod_name;
```

`1000`模式表示，只有匹配的列中（字段）含有1000即返回该行（记录）。

其他元字符如下：

```
. // 任意字符
| // 或,不过区别传统的|
[a-z1-9]    // 同传统的用法
[^a]    // 除却字符a
\\.     // 取消转义
[:alnum:]   // 任意字母或数字[a-zA-Z0-9]
[:alpha:]   // 任意字符 同[a-zA-z]
*   // 同{0,},匹配0个或多个
+   // 同{1,},1个或多个
?   // 同{0,1}
{n} // 匹配n次
{n,}// 匹配n次或以上
{n,m}// 匹配n次到m次
```

先前描述的区别还体现在下面一个例子:

```
select prod_name
from products
where prod_name REGEX '\\([0-9=]sticks?\\)'
order by prod_name;
```

这个的`sticks?`表示匹配单个字符s0次或1次。而不是先前的那种`stick`出现0次或1次。

`[:alpha:]`的用法同C++,如下:

```
select prod_name
from products
where prod_name REGEXP '[[:digit:]]{4}'
```

表示匹配连在一起的4位数字。

定位符

定位符同传统用法,即

```
^    文本的开始
$    文本的结尾
```

计算字段

什么是计算字段?

(关系型数据库RDB) 字段就是table中的列, 记录就是table中的行。计算字段是运行时在SELECT语句内创建的, 不实际存在于数据库的表中。

> 一般来说, 计算字段就是通过组合、格式化、转换实际存储的字段。这些"**计算操作**"可以在客户端完成, 但是一般在服务器端完成, 这样效率更高。这是DBMS的用处之一。

举例说明计算字段的用法。

concat()

拼接 (concatenate) 的缩写concat, 作为MySQL中的拼接关键词, 用作`Concat()`。更一般的, 在其他SQL语法中可以使用+或||, 这两者更为直观。比如

```
// string的+操作符用来连接两个string对象
string s=s1+s2
```

数学中使用||表示拼接。MySQL语法如下,

```
select concat(vend_name,' (',vend_country,')')
from vendors
order by vend_name;
```

`concat()`函数, 完成了拼接操作, 拼接了以下4个元素:

- 存储在vend_name字段中的名字 (实际的字段)
- 包含一个空格和一个(的串
- 存储在vend_country列中的国家
- 一个)

TRIM()

如名字rtrim, 该函数是为了删除字段右侧的多余空格。用法如下

```
select concat(rtrim(vend_name),' (',rtrim(vend_country),')')
from vendors
order by vend_name;
```

很显然,ltrim, 可以去除字段的左侧多余空格。trim, 可以去除两侧的多余空格。

别名 (alias) 或导出列 (derived column)

这里的别名和C++中的用处稍微不同。MySQL中的用法如下:

```
select concat(rtrim(vend_name),' (',rtrim(vend_country),')') as vend_title
from vendors
```

```
order by vend_name;
```

这时候，刚才的计算字段有了一个别名vend_title。这样客户端就可以引用该计算字段。这里的as指示SQL创建一个名为vend_title的计算字段。之后，任何客户端应用都可以按vend_title引用该字段，就像该计算字段实际存在于数据库中一样。（之后的章节就会介绍到可能会引用该计算字段的例子）。

该计算字段的别名vend_title有时候也成为导出列（derived column）。

计算字段的数值算术运算

计算字段可以完全其他计算操作，如数值运算。如下

```
select prod_id,
    quantity,
    iter_price,
    quantity*item_price as expanded_price -- quantity*iten_price的计算字段作为一个导出字段
expanded_price
from orderitems
where order_num=20005;
```

用于数据处理的函数

函数

如前所示的trim,rtrim,ltrim等都是处理数据的函数。

函数可以作用于字段（表中的列），也可以作用于临时的文本。如下：

```
select upper('hello'),lower('WORLD')
-- 以上两个计算字段也可以赋予别名
-- 用法如上，upper('hello') as FOO, lower('WORLD') as BAR
from products;
```

检索结果（显示结果如下）：

upper('hello')	lower('WORLD')
HELLO	world
HELLO	world
HELLO	world
...	...

> ! 应当注意函数没有sql语句的移植性强。每个DBMS的函数都不同或不支持。

MySQL中的函数如下，

文本处理函数

文本处理函数有如下，

```
rtrim(),ltrim(),trim()
upper()      // 文本转换为大写
lower()      // 文本转换为小写
length()     // 串的长度
其他文本函数不用记忆，用到再查
```

日期和时间处理函数

数据库中的日期和时间，它们的数据类型和格式都是按照一定的要求，以便可以快速的排序和过滤，并且节省物理存储空间。这个格式一般不被应用直接使用，需要“**计算**”处理一下。

下面直接给出例子介绍几个函数：

```
select cust_id,order_num
from orders
where order_date='2005-09-01';
```

这个检索可以检索出order_data字段值为`2005-09-01`的记录。而该保存日期的字段不一定是按照这个格式存储的。可能有如下的存储，

- 2005-09-01 11:30:05，此时上面的检索将会失败
- 或者其他格式存储的

MySQL日期处理函数的操作更为可靠：

```
select cust_id,order_num
from orders
where DATE(order_date)='2005-09-01';
```

其他相关函数：

```
time()  // 返回时间，如order_date字段存储的为2005-09-01 11:30:02格式，将会返回后面的时间
year()  // 返回日期中的年份
month()
day()
hour()
munite()
second()
```

几个无参数的函数：

```
curdate()   // 无参数,返回当前日期
curtime()
```

进阶的例子

如果想检索某两个日期期间的记录：

```
select cust_id,order_num
from orders
where date(order_date) between '2005-09-01' and '2005-05-30';
-- 或者
where year(order_date)=2005 and month(order_date)=9;
```

这里的BETWEEN操作符用来匹配日期的范围。

数值处理函数

在各类函数中，各个DBMS都有相同的数值处理函数：

```
abs()
cos()
exp()   // 指数值
mod()   // 余数
pi()    // 返回圆周率,无参数
rand()  // 返回随机数
sin()
tan()
sqrt()
```

汇总数据

SQL的进行数据汇总也是利用的函数-聚集函数。

聚集函数 (aggregate function)

举几个函数就明白聚集函数的用处了。

```
avg()   // 返回某个字段的平均值
count() // 返回某个字段的记录数,即某个列的行数
max()   // 返回某个字段的最大值
min()
sum()
```

> NULL值的行（记录）将会被忽略。因为NULL没有数据类型，无法参与计算，它不是0。

AVG()

```
select avg(prod_price) as avg_price
from products;
```

更进一步，我们只计算满足给定要求的记录（行）的平均值。

```
select avg(prod_price) as avg_price
from products
where vend_id =1003;
```

COUNT()

> NULL值的行也**可能**会被count，因为NULL值不影响该计算。

两种用法：

- `count(*)`，计算整个表的行数，计算包括NULL值的行
- `count(column)`，计算某个字段（列）的行数，忽略NULL值
  可以看出*的作用特点。

MAX(),MIN()

`max()`为一个参数，即某个字段，返回该字段记录的最大值。**NULL**在这里很显然会被忽略。

```
select max(prod_price) as max_price
from products;
```

> `max()`可以应用于非数值的字段，只要该字段可以排序。

`min()`不再赘述。

SUM()

`sum()`函数可以作用于计算字段（**其他所有聚集函数都可以作用于计算字段**）。如

```
select sum(item_price*quantity) as total_price
from orderitems
where order_num=2005;
```

> 同样，NULL值将被忽略

类似的我们也可以应用于这样的“计算字段”：

```
select avg(distinct prod_price) as avg_price
from products
where vend_id=1003;
```

组合聚集函数

如果我们将聚集函数组合起来呢？

```
select count(*) as num_items,
        min(prod_price) as price_min,
        max(...) as ...,
        avg(...) as ...
from products;
```

检索结果将会如下所示：

num_items	price_min	...	...
...	...	...	...

分组数据

分组数据，是为了能够汇总表中的子集。涉及两个select语句子句：

- GROUP BY
- HAVING

数据分组

分组允许把数据分为多个逻辑组，以便对每个组进行聚集计算。

创建分组

这里用一个例子理解分组：

```
select vend_id,count(*) as num_prods
from products
group by vend_id;
```

检索结果为：

vend_id	num_prods

vend_id	num_prods
1001	3
1002	2
1003	7
1005	2

这里的GROUP BY子句指示MySQL按vend_id排序并分组数据，这导致对每个vend_id而不是整个表计算num_prods一次。检索结果可以看出，vend_id=1001的有3个产品，依次类推。

分完组之后，select语句会对每个组进行一次计算。这些操作是系统自动完成的。

GROUP BY子句可以：

- 可以嵌套
- NULL值作为一个一分组返回
- 跟在where之后，Order by之前

过滤分组

过滤分组，即包括哪些组，排除哪些组。

where子句过滤行（记录），不可以过滤分组。过滤分组使用HAVING子句。因此，HAVING子句类似于where，作用于分组。并且，HAVING可以代替where。（每个行看成一个分组?）

```
select cust_id,count(*) as orders
from orders
group by cust_id    -- 依cust_id进行排序分组
having count(*) >=2;    -- having子句过滤分组，依count(*)是否大于等于2
```

> where和having子句的差别：where子句对行进行过滤，作用在分组前（group by子句前），having可以对组进行过滤，作用于分组之后。where先过滤行，这些被过滤掉的行不会出现在任何分组中。

```
select vend_id,count(*) as num_prods
from products
where prod_price>=10    -- 先对行进行过滤，这里的where也可以替换为having
group by vend_id
having count(*)>=2;
```

分组和排序

这里看一下group by和order by的区别：

order by	group by

order by	group by
排序检索结果	排序行并进行分组（但是这里的排序不是SQL规范，依赖于具体的DBMS）
任意字段都可以使用（不包括计算字段）	只能选择列或表达式列（计算字段），而且必须使用每个选择列表达式

```
select order_num,sum(quantity*item_price) as ordertotal
from orderitems
group by order_num
having sum(quantity*item_price)>=50
order by ordertotal;     -- 这里的order by 必须是表的字段
```

这个就是先使用group by依照order_num进行分组（可能排序也可能不排序），然后having对分组进行过滤，最后order by对检索结果按照ordertotal进行排序。

> 这里注意：where子句中的字段（列）必须要匹配，字段匹配，字段数目也要匹配。

使用计算字段作为子查询

这里给出一个精确的例子，假如需要显示customers表中的每个客户的订单总数。订单与相应的客户ID存储在orders表中。为了执行这个操作，需要：

1. 从customer中检索出客户列表
2. 对于1中返回的每个记录，即每个客户，统计其在orders中的订单数目。

可以使用计算字段作为子查询

```
select count(*) as orders from orders where cust_id=10001;  -- 对客户ID为10001的订单进行count

-- 对每个客户执行count(*)计算，将count(*)作为一个子查询

select cust_name,cust_state,
       (select count(*) from order where orders.cust_id=customers.cust_id) as orders
from customers
order by cust_name;
```

这里的SQL语句，对customer表中的每个客户返回3列：cust_name,cust_state,orders。orders是一个计算字段，该计算字段由select 子查询建立。该子查询对检索出的每个客户执行一次。

> 相关子查询（correlated subquery）涉及外部查询的子查询

涉及外部查询的时候，最好使用完全限定的字段。即应该显示指明该字段属于哪个表中的字段。

```
select cust_name,cust_state,(select count(*) from order where cust_id=cust_id) as
orders
from customers
order by cust_name;
```

以上语句显然会有歧义。此时的`where cust_id=cust_ie`都是指的custmers中的cust_id字段。此时总是返回orders表中的订单总数。