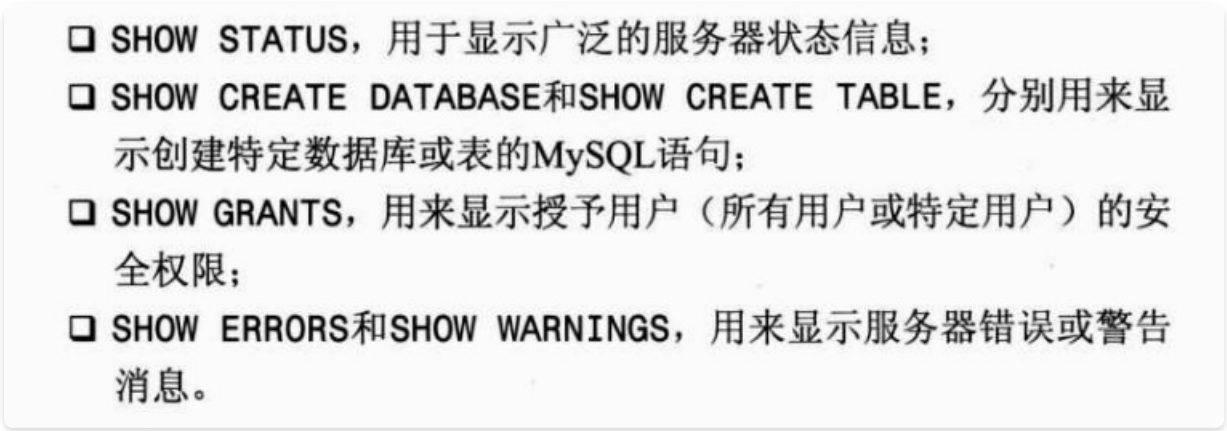
mysql.md

小書匠

目录

我们可以通过show columns from 表名;来查看列表。
在课堂上使用desc + 表名来达到同样的效果



enter description here

SELECT语句(重要)

在Mysql中多条语句可以一起书写
例如:use xscj;select * from tb_student;

在使用select时我们会发现检索出来的值很多都是重复的,如果你不想要那些重复的值可以使用distinct

我们可以通过limit来限制返回几行数据
例如:select * from tb_student limit 5;说明只显示五行数据
而select * from tb_student limit 5,5;说明返回从第五行开始后面的五行数据

注意行是从0开始计数的

排序检索数据

通常我们可以使用order by让返回的数据呈现出一个自己希望的顺序
如果你不想正序而想反序,则可以使用desc。正序是asc,当然!我们绝大部分都不会使用asc,因为这是默认的!

我们可以使用order by和limit的组合来找出一个列中的最大值和最小值

过滤数据(whereBETWEEN AND)

通常我们使用where语句可以精确筛选数据,如果有order by语句的话order by要放在where语句后面,不然会出错

操作符	说明
=	等于

操作符	说明
<>	不等于
!=	不等于
<	小于
<=	小于等于
>	大于
>=	大于等于
BETWEEN	在指定的两个值之间

我们最应该注意的是BETWEEN这个操作符

例子:select * from tb_student where s_num between 100 and 200;

从这里我们可以看出,在使用BETWEEN时必需指定两个值--所需范围的最低值和最高值.并且这两个值需要使用AND连接起来

在where语句中有一个特殊的字句专门用来检查null值,这就是IS NUll字句

数据过滤(where and or not)

where语句支持and和or实现更为复杂的查找(如果and和or过多,推荐使用括号来区分谁先执行,毕竟在mysql中最优先执行and其次是or)

where语句还支持in操作符,但是注意IN不是指范围,而是指括号里面的值

```
select * from tb_student where s_num in(101,201);-- 不是指范围,而是指s_num等于101或者102的
```

or能完成的操作in也能完成,他们就像while和for循环一样

not操作符用来否定它后面所跟的任何条件,not一定要和in组合使用,而且in一定要使用括号,不然会无效

```
select * from tb_student where s_num not in(101); -- 哪怕只有一个值也必须加括号!
```

用通配符(like %)进行过滤

在我们需要模糊查找时则必须使用通配符 like
%表示任何字符出现的任意次数(有点像正则表达式中的*)
注意,虽然%很强大,但是它还是不能匹配一个东西----NULL

```
select * from tb_student where s_name like 'a%';
```

_(下划线)在mysql里匹配任意一个字符(有点像正则表达式中的.)

使用正则表达式查询(REGEXP)

```
select * from tb_student where s_num regexp '101';
```

创建计算字段

在实际开发中, 如果我们需要将两个列的数据拼接起来可以使用Concat()函数

```
select concat(s_num,s_name) from tb_student; --可以是两个以上
```

在实际开发中,如果需要删除右侧多余空格可以使用RTrim()函数,左侧可以使用LTrim(),Trim()中间

```
select concat(s_num,s_name,RTrim(s_note)) from tb_student;
```

起别名通常使用AS

mysql还支持算数运算,+-*/这些简单的计算都能够支持

```
select (s_num * total_credit) as total from tb_student where s_num = 102;
```

使用简单函数来处理数据

文本函数

注意, 函数没有sql的移植性强, 所以不要过度使用函数

我们可以使用UPPER()来让小写字体变为大写

```
select UPPER(s_name) from tb_student;
```

函数	说明
Left()	返回左边的字符
lenght()	返回字符串的长度
locate()	找出当前串的字串
Lower()	将当前串转换为小写
LTRIM()	去掉左侧的空格

函数	说明
RTrim()	去掉右侧的空格
Right()	返回串右边的字符
soundex()	返回串的SOUNDEX的值
SubString()	返回字串的字符
Upper()	将串转换为大写

日期函数

感觉很复杂又不经常用, 先不看

数值处理函数

函数	说明
Abs()	返回一个数的绝对值
Cos()	返回一个角度的余弦
Exp()	返回一个数的指数值
Mod()	返回操作数的余数
Pi()	返回圆周率
Rand()	返回一个随机数
Sin()	返回一个角度的正弦
Sqrt()	返回一个数的平方根
Tan()	返回一个角度的正切

汇总数据(聚合函数)

函数	说明
AVG()	返回某列的平均值
COUNT()	返回某列的行数
MAX()	返回某列的最大值
MIN()	返回某列的最小值
SUM()	返回某列的数值之和

AVG(),MAX(),MIN(),SUM()函数会忽略列值为NULL的行
COUNT()函数如果指定列名,则指定列为NUll时忽略,但如果时count(*)时则不忽略

需要不同的值吗?distinct会满足你的要求

```
select avg(distinct s_num) from tb_student;
```

分组数据(GROUP BY,HAVING)

```
select count(*),s_zhuanye from tb_student
group by s_zhuanye;
```

使用WITH ROLLUP关键字可以得到每个分组汇总的总级别的值

```
select count(*),s_zhuanye from tb_student
group by s_zhuanye WITH ROLLUP;
```

如果你想过滤分组的话,HAVING关键词可以帮助你(WHERE过滤行,HAVING过滤列)

```
select count(*),s_zhuanye from tb_student
group by s_zhuanye HAVING count(*) > 3;
```

where在数据分组前进行过滤,而having在数组分组后进行过滤

字句	说明	是否必须使用
SELECT	要返回的列或表达式	是
FROM	从中检索数据的表	仅在选择表数据时使用
WHERE	行级过滤	否
GROUP BY	分组过滤	仅在按组计算聚集时使用
HAVING	组级过滤	否
ORDER BY	输出排序顺序	否
LIMIT	要检索的行数	

使用子查询

注意:在使用子查询时尽量分成多行显示,特别是很复杂的子查询,这样能极大简化子查询的使用

```
select total_credit from tb_student where s_num in (select s_num from tb_student where s_num > 200); -- 不要这样一行写!!
```

```
SELECT
 total_credit
FROM
 tb_student
WHERE
 s_num IN ( SELECT s_num FROM tb_student WHERE s_num > 200 ); --  很漂亮的代码!
```

联接表

```
SELECT
 vend_name,
 prod_name,
 prod_price
FROM
 vendors,
 products
WHERE
 vendors.vend_id = products.vend_id
ORDER BY
 vend_name,
 prod_name;
```

如果没有通过where连接两个表则返回的是笛卡尔积,检索出的行数将是第一个表的行数乘以第二个表的行数

```
SELECT
 vend_name,
 prod_name,
 prod_price
FROM
 vendors,
 products
ORDER BY
 vend_name,
 prod_name;
```

我们还可以使用sql推荐的INNER JOIN方法

```
SELECT
 vend_name,
 prod_name,
 prod_price
FROM
 vendors
 INNER JOIN products ON vendors.vend_id = products.vend_id;
```

我们还可以连接多个表,但是注意,表连接的越多性能越低,所以要尽最大可能少连接表

```
SELECT prod_name,vend_name,prod_price,quantity
from orderitems,products,vendors
WHERE products.vend_id = vendors.vend_id
AND orderitems.prod_id = products.prod_id
and order_num = 20005;
```

创建高级连接

在sql语句种, 我们可以给表起别名, 这样做有两种好处:1缩短sql语句2允许在单挑select语句中多次使用相同的表

```
select * from tb_student as student where student.s_num = 101;
```

我们应该多使用起别名的方式来替代子查询,因为起别名比子查询快得多!

组合查询(UNION)

在sql中我们可以使用UNION关键字来结合两个select语句(注意:重复的数据UNION会只保留一个),但是我们也可以通过UNION ALL来保留所有数据,包括重复的

```
select * from tb_student where s_num <105
UNION
select * from tb_student where s_num >= 200;

select * from tb_student where s_num <105
UNION ALL
select * from tb_student where s_num >= 200; -- 保留重复的数据
```

使用UNION的主要用途是减少where语句,同时增加一个where永远也不能完成的重复数据保留(UNION ALL)

全文本搜索

注意!!!:不是所有的引擎都支持全文本搜索,我们最常用的两个搜索引擎MyISAM和InnoDB,前者支持全本文搜索,但后者不支持全文本搜索!一定要记住这一点!!!

我们通过在创建表时指定FULLTExt和最后设置引擎来启动全文本搜索

```
CREATE TABLE productnotes(
note_id int not null auto_increment,
prod_id char(10) not null,
note_date datetime not null,
note_text text null,
PRIMARY key(note_id),
FULLTEXT(note_text))engine=MyISAM;
```

在开启全文本搜索后,我们通过Match()来指定被搜索的列,Against()来指定要使用的搜索表达式.我们也可以通过使用BINARY关键字来指定区分大小写,如果不指定BINARY关键字则不区分大小写

```
SELECT note_text
FROM productnotes
WHERE MATCH(note_text) Against('rabbit');
```

我们也可以使用扩展查询(WITH QUERY EXPANSION),它会查找和第一行类似的值,并按顺序返回

```
SELECT note_text
from productnotes
```

```
WHERE MATCH(note_text) Against('anvils' WITH QUERY expansion);
```

在全文查询中还支持布尔文本搜索,这是一种很厉害的搜索方式

具体细节如下:

1. 要匹配的值
2. 要排斥的值(如果某行包含这个值,则不返回该行,即使它包含其他指定的词也是如此)
3. 排列提示(指定某些词比其他词更重要,更重要的词等级更高)
4. 排列式分组
5. 另外一些内容
6. **没有FULLTEXT也可以使用(但是速度异常的低)**

布尔操作符	说明
+	包含,词必须存在
-	排除,词必须不存在
>	包含,而且增加等级值
<	包含,而且减少等级值
()	把词组成子表达式(允许这些子表达式作为一个组被包含,排除,排列等)
-	取消一个词的排序值
*	词尾的通配符
""	定义一个短语(与单个词的列表不一样,它匹配整个短语以便排除或包含这个短语

```
SELECT note_text
from productnotes
WHERE MATCH(note_text) against('heavy -rope*' in boolean mode);
```

下面举几个例子，说明某些操作符如何使用：

输入

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('+rabbit +bait' IN BOOLEAN MODE);
```

173

分析 这个搜索匹配包含词rabbit和bait的行。

输入

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('rabbit bait' IN BOOLEAN MODE);
```

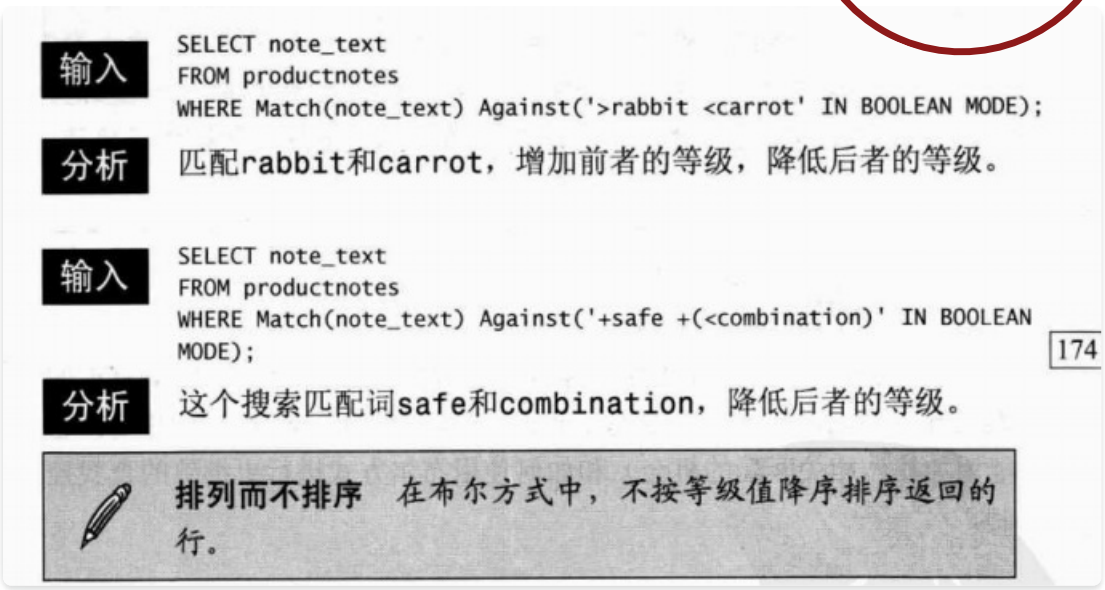
分析 没有指定操作符，这个搜索匹配包含rabbit和bait中的至少一个词的行。

输入

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('"rabbit bait"' IN BOOLEAN MODE);
```

分析 这个搜索匹配短语rabbit bait而不是匹配两个词rabbit和bait。

enter description here



输入

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('>rabbit <carrot' IN BOOLEAN MODE);
```

分析 匹配rabbit和carrot，增加前者的等级，降低后者的等级。

输入

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('+safe +(<combination)' IN BOOLEAN
MODE);
```

174

分析 这个搜索匹配词safe和combination，降低后者的等级。

排列而不排序 在布尔方式中，不按等级值降序排序返回的行。

enter description here

注意:许多词出现很高,搜索它们没有任何用处(返回太多结果了),因此mysql规定一个词出现在50%以上的行中,则将它作为一个非用词忽略.但是这个规则不适用与IN BOOLEAN MODE

如果表中的行少于3行,则全文本搜索不返回任何结果

忽略词中的单引号,例如don't索引为dont

仅在MyISAM引擎支持全文搜索

插入数据(INSERT)

```
insert into 表名() values(); -- 插入整行数据的简单例子
```

虽然上面的例子看起来很简单,但是非常不安全,我们应该尽量少使用!!(因为高度依赖与表的顺序,如果可以尽量使用下面这种)

```
insert into tb_student(s_num,s_name,s_note) values(); --这样是推荐的做法
```

我们还可以使用insert select语法来插入检索的数据

```
insert into tb_student()
select * from tb_student;
```

更新和删除数据(UPDATE和DELETE)

UPDATE语句由三部分组成

1. 要更行的表
2. 列名和他们的新值
3. 确定要更新的行和过滤条件

在使用UPDATE中如果有错误发生则之前的修改会被回滚到最初状态,如果你想即使发生错误,也继续更行则可以使用UPDATE IGNORE

如果想从表中删除所有的行不要使用DELETE,可以使用TRUNCATE DELETE语句,它可以完成相同的工作,但速度快得多

小心使用UPDATE和DELETE,因为mysql没有撤回按钮

创建和操纵表

创建一张表必须给出以下信息:

1. 新表的名字, 在关键词create table之后给出
2. 表列的名字和定义, 用逗号分割

```
CREATE TABLE demo ( id INT ( 100 ), NAME VARCHAR ( 10 ), address VARCHAR ( 100 ), birth_day VARCHAR ( 100 )
);
```

在创建表时可以使用NULL NOT NULL,PRIMARY KEY AUTO_INCREMENT(每个表只允许有一个AUTO_INCREMENT列, 而且它必须被索引[比如使他成为主键]) DEFAULT(指定默认值)

我们还可以在创建表的最后加上存储引擎类型

```
CREATE TABLE demo ( id INT ( 100 ), NAME VARCHAR ( 10 ), address VARCHAR ( 100 ), birth_day VARCHAR ( 100 )
)ENGINE=Innodb;
```

常见的存储引擎:
1. Innodb:一个可靠的事务处理引擎,它不支持全文本搜索
2. MEMORY:在功能上等同于MyISAM,但由于数据存储在内存中(不是磁盘),所以速度很快(特别适合临时表)
3. MyISAM:一个性能极高的引擎,它支持全文本搜索,但不支持事务处理

在这里有一个很可怕的特点,混用引擎有一个大缺陷,那就是外键不能跨引擎!!!

为了更行表的定义(不是更新表的数据)可以使用ALTER TABLE

```
alter table tb_student add name varchar(20); -- 向tb_student表中加入 name列
alter table tb_student drop column name;tb_student表中删除name列
```

其实alter table还有一种常见的用处,那就是定义外键

删除表(而不是表里的内容,可以使用drop table

可以使用rename table来重命名一张表

```
rename table tb_student to student;
```

使用视图

视图是一张虚拟的表, 与包含数据的表不一样, 视图只包含使用时动态检索数据的查询

为什么使用视图:
1. 重用sql语句
2. 简化复杂的sql操作,在编写查询后,可以方便地重用它而不必知道它的基本查询细节
3. 使用表的组成部分而不是整个表
4. 保护数据,可以给用户授予表的特定部分访问权限而不是整个表的访问权限
5. 更改数据格式和表示,视图可返回与底层表的表示和格式不同的数据

如果你使用多个联接和过滤创建了复杂的视图或者嵌套了视图,性能就会下降的很厉害

视图的规则和限制:
1. 与表一样,视图必须唯一命名(不能给视图取与别的视图或表相同的名字)
2. 对于可以创建的视图数目没有限制
3. 为了创建视图,必须有足够的访问权限,这些限制通常由数据库管理人员赋予
4. 视图可以嵌套,即可以利用从其他视图中检索的数据的查询来构造一个视图
5. ORDER BY可以用在视图中,但如果从该视图检索数据SELECT中也包含ORDER BY,那么该视图中的ORDER BY被覆盖
6. 视图不能索引,也不能有关联的触发器或默认值
7. 视图可以和表一起使用,列入编写一条联结表和视图的SELECT语句

通常使用select view来创建视图

- 视图用CREATE VIEW语句来创建。
- 使用SHOW CREATE VIEW viewname；来查看创建视图的语句。
- 用DROP删除视图，其语法为DROP VIEW viewname;。
- 更新视图时，可以先用DROP再用CREATE，也可以直接用CREATE OR REPLACE VIEW。如果要更新的视图不存在，则第2条更新语句会创建一个视图；如果要更新的视图存在，则第2条更新语句会替换原有视图。

enter description here

```
CREATE view my_view as SELECT * from tb_student; -- 创建视图的方式
```

```
CREATE VIEW student_view AS SELECT
grade,
tb_student.s_num,
s_name
FROM
 tb_student,
 tb_sc
WHERE
 tb_student.s_num = tb_sc.s_num; -- 这样创建视图可以极大方便我们之后的查询
```

我们也可以通过视图更新父表(但是如果视图定义了以下几种操作就不行了

1. 分组(使用group by和having)
2. 联接
3. 子查询
4. 并
5. 聚集函数(Min(),count(),Sum()等)
6. Distinct
7. 导出(计算)列

视图主要用来方便检索数据,尽最大可能不要拿来更新数据

使用存储过程

存储过程简单来说就是为了以后的使用而保存的一条或多条mysql语句的集合,可将它们视为批文件

为什么要使用存储过程:

1. 通过把处理封装在容易使用的单元中,简化复炸的操作.
2. 由于不要求反复建立一系列的处理步骤,保证了数据的完整性
3. 简化对变动的管理,如果表名,列名或业务逻辑有变化,只要更改存储过程的代码就可以了(提高了安全性)
4. 提高性能

创建存储过程

我们可以通过call命令来执行调用存储过程

使用游标

游标是用select检索出来的结果集,它可以很方便的查看数据的第一行,后十行之类的

使用游标的几个过程:

- 在能够使用游标前,必须定义它,在这个过程没有检索数据,它只是定义要使用的select语句

- 一旦申明后,必须打开游标以供使用,这个过程用前面定义的select语句把实际检索出来
- 对于填有数据的游标,根据需要取出(检索)各行
- 在结束游标使用时,必须关闭游标

使用触发器

如果我们想让一条sql命令在特定时间执行就需要触发器

在以下语句中可以启动触发器:
1. DELETE
2. INSERT
3. UPDATE

在创建触发器时需要给出4条信息:
1. 唯一的触发器名
2. 触发器关联的表
3. 触发器应该响应的活动(DELETE,INSERT或UPDATE)
4. 触发器何时执行(处理之前还是之后)

我们可以使用create trigger语句来创建一个简单的触发器

只有表才支持触发器,视图不支持(临时表也不支持)

每个表每个事件只允许定义一个触发器,因此一个表最多支持6个触发器(每条SELECT,INSERT,DELETE的前后),单一触发器不能和多个时间或多张表关联

如果BEFORE触发器失败,则mysql将不执行请求的操作,此外如果BEFORE触发器或者语句本身失败,则mysql将不执行after触发器

我们可以使用drop trigger语句来删除触发器(触发器不能更改,所以为了修改一个触发器必须要先删除再重新创建

管理事务处理

我们通常使用COMMIT和ROLLBACK语句来管理事务处理

并不是所有的存储引擎都支持事务处理, MyISAM不支持, 而Innodb支持

事务处理可以用来维护mysql数据库的完整性,它保证成批的mysql操作要么完全执行,要么完全不执行

安全管理

我们可以使用create user来创建一个新用户

```
create user lindaxia IDENTIFIED by 'toor';
```

我们可以使用rename user来更改一个用户

```
rename user lindaxia to linguangfu;
```

我们可以使用drop user来删除一个用户

```
drop user 用户名
```

设置访问权限

查看账号的权限可以使用SHOW GRANTS FOR

```
show GRANTS FOR lindaxia;
```

设置权限通过grant,但是你需要给出以下几个信息

1. 要授予的权限
2. 要被授予权限的数据库或表
3. 用户名

```
GRANT SELECT ON -- 授予用户select权限
 xscj.* TO lindaxia; -- 授予的数据库是xscj,表是所有的表,授予的用户是lindaxia
```

我们可以通过revoke来撤销给用户的权限

```
revoke select on xscj.* from lindaxia; -- 这条操作必须存在,不然就会撤销失败
```

如果我们忘记了之前赋予用户的命令了怎么办?没关系,我们可以通过show grants来查看

```
show grants for lindaxia;
```

我们可以通过set password来更改密码(在不指定用户名时默认修改当前用户的密码)

```
set PASSWORD for lindaxia = PASSWORD('root');
```

数据库维护

在备份前最好使用一句flush tables来保证数据全部刷新进了数据库

我们通常使用ANALYZE TABLE来检查表键是否正确

```
ANALYZE TABLE tb_student;
```