MYSQL 必知必会笔记 (未完)

作者：someone38063
原文链接：https://ld246.com/article/1562409366141
来源网站：链滴

什么是数据库

数据库是一个以某种有组织的方式存储的数据集合。

数据库(database)：保存有组织的数据的容器（通常是一个文件或一组文件）。

表

表是一种结构化的文件，可用来存储某种特定类型的数据。

表(table)：某种特定类型数据的结构化清单。

- 数据库中的每个表都有一个名字，用来标识自己。 **此名字是唯一的**，这表示数据库中没有其他表具有相同的名字。

虽然在相同数据库中不能两次使用相同的表名，但在不同的数据库中却可以使用相同的表名

模式(schema)

关于数据库和表的布局特性的信息

列和数据类型

表由列组成。列中存储着表中某部分的信息

列(column)：表中的一个字段。所有的表都是由一个或多个列组成的

- 数据库中每个列都有相应的数据类型。数据类型定义了列可以存储的数据种类

数据类型(datatype)

所容许的数据的类型。每个表列都有相应的数据类型。他限制(或容许)该列中存储的数据

- 数据类型还帮助正确的排序数组，并在优化磁盘使用方面起重要作用

行

表中的数据是按行存储的，所保存的每个记录存储在自己的行内。

行(row)：表中的一个记录

经常听到行(row)时称其为数据库记录(record)。很大程度上这两个术语是可以相互代替的，但从技术说，行才是正确的术语

主键

主键(primary key)：一列(或一组列)，其值能够唯一区分表中每个行

唯一标识表中每行的这个列(或这组列)成为主键。主键用来表示每一个特定的行。

没有主键，更新或删除表中的特定行很困难，因为没有安全方法保证只涉及相关的行

● 应该总是定义主键

虽然并不总是都需要主键，但大多数数据库设计人员都应保证他们创建的每个表中具有一个主键，以于操作和管理

● 表中的任何列都可以作为主键，只要它满足以下条件

1. 任意两行不具有相同的主键值
2. 每个行都必须具有一个主键值(主键列不允许NULL值)

主键通常定义在表的一列上，但这并不是必须的，也可以一起使用多个列作为主键。在使用多列作为键时，上述条件必须应用到构成主键的所有列，所有列值的组合必须是唯一的(单个列的值可以不唯一)

客户机-服务器软件

● 与数据文件打交道的只有服务器软件。关于数据、数据添加、删除和数据更新的所有请求都由服务软件完成。这些请求或更改来自运行客户机软件的计算机。客户机是与用户打交道的软件。例如，如你请求一个按字母顺序列出的产品表，则客户机软件通过网络提交该请求给服务器软件。服务器软件理这个请求，根据需要过滤、丢弃和排序数据；然后把结果送回到你的客户机软件。

● 所有这些活动对用户都是透明的。数据存储在别的地方，或者数据

库服务器为你完成这个处理这一事实是隐藏的。你不需要直接访问数据

文件。事实上，多数网络的建立使用户不具有对数据的访问权，甚至不

具有对存储数据的驱动器的访问权。

MYSQL命令行实用程序

注意：

1. 命令输入在mysql>之后
2. 命令用;或\g结束，换句话说，仅按Enter不执行命令
3. 输入help或\h获得帮助；例如：select help 获得使用select语句的帮助
4. 输入quit或exit退出命令行

MYSQL Administrator

是一个图形交互客户机，用来简化MYSQL服务器的管理

MYSQL Query Browser

为一个图形交互客户机，用来编写和执行MYSQL命令

使用MYSQL

连接

为了连接到MySQL，需要以下信息：

1. 主机名（计算机名）——如果连接到本地MySQL服务器，为 localhost；
2. 端口（如果使用默认端口3306之外的端口）；
3. 一个合法的用户名；
4. 用户口令（如果需要）。

例如mysql -h localhost -P 3396 -u root -p 123

选择数据库

在你最初连接到MySQL时，没有任何数据库打开供你使用。在你能执行任意数据库操作前，需要选一个数据库。为此,可使用USE关键字。

关键字(key word) 作为MySQL语言组成部分的一个保留字。决不要用关键字命名一个表或列。

例如：为了使用username数据库，

```
mysql>use username;
Database changed
```

● USE语句并不返回任何结果。依赖于使用的客户机，显示某种形式的通知。例如，这里显示出的 Database changed 消息是mysql命令行实用程序在数据库选择成功后显示的。

必须先使用 USE 打开数据库，才能读取其中的数据。

了解数据库和表

● SHOW DATABASES; 返回可用数据库的一个列表。包含在这个列表中的可能是MySQL内部使用的据库（如例子中的 mysql 和information_schema ）。当然，你自己的数据库列表可能看上去与这的不一样。

● 为了获得一个数据库内的表的列表，使用 SHOW TABLES;

● SHOW 也可以用来显示表列： show columns from [表名];

```
+----------+--------------+------+-----+-------------------+-----------------------------+
| Field    | Type         | Null | Key | Default           | Extra                       |
+----------+--------------+------+-----+-------------------+-----------------------------+
| variable | varchar(128) | NO   | PRI | NULL              |                             |
| value    | varchar(128) | YES  |     | NULL              |                             |
| set_time | timestamp    | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
|
| set_by   | varchar(128) | YES  |     | NULL              |                             |
+----------+--------------+------+-----+-------------------+-----------------------------+
```

● SHOW COLUMNS 要求给出一个表名（这个例子中的 FROMcustomers ），它对每个字段返回一，行中包含字段名、数据类型、是否允许 NULL 、键信息、默认值以及其他信息（如字段set_time的n update CURRENT_TIMESTAMP ）。

● describe语句

MySQL支持用 DESCRIBE 作为 SHOW COLUMNSFROM 的一种快捷方式。换句话说DESCRIBE customers; 是SHOW COLUMNS FROM customers; 的一种快捷方式。

所支持的其他show语句：

1. SHOW STATUS ，用于显示广泛的服务器状态信息；

2. SHOW CREATE DATABASE和SHOW CREATE TABLE ，分别用来显示创建特定数据库或表的MyQL语句；

3. SHOW GRANTS ，用来显示授予用户（所有用户或特定用户）的安全权限；

4. SHOW ERRORS 和 SHOW WARNINGS ，用来显示服务器错误或警告消息。

MYSQL 5的新增内容：MySQL 5支持一个新的 INFORMATION_SCHEMA 命令，可用它来获得和过模式信息。

检索数据

SELECT 语句

为了使用 SELECT 检索表数据，必须至少给出两条信息——想选择什么，以及从什么地方选择。

检索单个列

select name from products;:利用SELECT语句从products表中检索一个名为name的列。所需的列在SELECT关键字之后给出,FROM关键字指出从其中检索数据的表名。

这一条简单 SELECT 语句将返回表中所有行。数据没有过滤（过滤将得出结果集的一个子集），也没排序。

- 结束SQL语句

多条SQL语句必须以分号(;)分隔。MySQL如同多数DBMS一样，不需要在单条SQL语句后加分号。但定的DBMS可能必须在单条SQL语句后加上分号。当然，如果愿意可以总是加上分号。事实上，即使一定需要，但加上分号肯定没有坏处。如果你使用的是 mysql命令行，必须加上分号来结束 SQL 语。

- SQL语句和大小写

请注意，SQL语句不区分大小写，因此SELECT 与 select 是相同的。同样，写成 Select 也没有关系许多SQL开发人员喜欢对所有SQL关键字使用大写，而对所有列和表名使用小写，这样做使代码更易阅读和调试。最佳方式是按照大小写的惯例，且使用时保持一致。

- 使用空格

在处理SQL语句时，其中所有空格都被忽略。SQL语句可以在一行上给出，也可以分成许多行。多数SL开发人员认为将SQL语句分成多行更容易阅读和调试。

检索多个列

要想从一个表中检索多个列，使用相同的 SELECT 语句。唯一的不同是必须在 SELECT 关键字后给出个列名，列名之间必须以逗号分隔。

- 当心逗号

在选择多个列时，一定要在列名之间加上逗号，但最后一个列名后不加。如果在最后一个列名后加了号，将出现错误。

例如select username,password,id from products;

检索所有列

除了指定所需的列外（如上所述，一个或多个列）， SELECT 语句还可以检索所有的列而不必逐个列它们。这可以通过在实际列名的位置使用星号（ * ）通配符来达到，例如：select * from products;

- 如果给定一个通配符（ * ），则返回表中所有列。列的顺序一般是列在表定义中出现的顺序。但有候并不是这样的，表的模式的变化（如添加或删除列）可能会导致顺序的变化。
- 检索未知列

使用通配符有一个大优点。由于不明确指定列名（因为星号检索每个列），所以能检索出名字未知的。

检索不同的行

假设select语句返回了14行，但是不同的值只有4个，想要检索出不同值的列表：使用DISTINCT关键，顾名思义，此关键字指示MySQL只返回不同的值

- SELECT DISTINCT id 告诉MySQL只返回不同（唯一）的id 行，因此只返回4行；如果使用 DISTINCT关键字，它必须直接放在列名的前面。

限制结果

SELECT 语句返回所有匹配的行，它们可能是指定表中的每个行。为了返回第一行或前几行，可使用LIMIT子句：select username from products limit 5;

此语句使用 SELECT 语句检索单个列。 LIMIT 5 指MySQL返回不多于5行

- 为得出下一个5行，可指定要检索的开始行和行数： select username from products limit 5,5;

LIMIT 5, 5 指示MySQL返回从行5开始的5行。第一个数为开始位置，第二个数为要检索的行数。

所以，带**一个值**的LIMIT总是从**第一行**开始，给出的数为返回的行数。带**两个值**的LIMIT可以指定从行为第一个值的位置开始。

- 行 0

检索出来的第一行为行0而不是行1。因此，LIMIT 1,1将检索出第二行而不是第一行。

- 在行数不够时

LIMIT中指定要检索的行数为检索的最大行数。如果没有足够的行（例如，给出 LIMIT 10, 5 ，但只有3行），MySQL将只返回它能返回的那么多行。

- MySQL 5的 LIMIT语法

  - LIMIT 3, 4 的含义是从行4开始的3行还是从行3开始的4行？如前所述，它的意思是从行3开始的行，这容易把人搞糊涂。
  - 由于这个原因，MySQL 5支持 LIMIT 的另一种替代语法。 LIMIT4 OFFSET 3意为从行3开始取行，就像LIMIT 3, 4一样。

使用完全限定的表名

之前使用的SQL例子只通过列名引用列。也可能会使用完全限定的名字来引用列（同时使用表名和列）：[表名].[列]。例如：select products.username from products;表名也可以是完全限定的：[库名]

[表名]

排序检索数据

排序数据

使用select username from products;检索出的数据并不是以纯粹的随即顺序显示的，如果不排序，据一般将以它在底层表中出现的顺序显示。这可以是数据最初添加到表中的顺序。但是，如果数据后进行过更新或删除，则此顺序将会受到MySQL重用回收存储空间的影响。因此，如果不明确控制的，不能（也不应该）依赖该排序顺序。关系数据库设计理论认为，如果不明确规定排序顺序，则不应假定检索出的数据的顺序有意义。

子句（clause）

SQL语句由子句构成，有些子句是必需的，而有的是可选的。一个子句通常由一个关键字和所提供的据组成。子句的例子有SELECT语句的FROM子句，

- 为了明确地排序用 SELECT 语句检索出的数据，可使用 ORDER BY 子句。ORDER BY 子句取一个多个列的名字，据此对输出进行排序。

select username from products order by username;

按多个列排序

下面的代码检索3个列，并按其中两个列对结果进行排序:首先按price，然后再按username排序

select username,price,id from products order by price,username

重要的是理解在按多个列排序时，排序完全按所规定的顺序进行。

换句话说，对于上述例子中的输出，仅在多个行具有相同的price值时才对产品按username进行排序如果price列中所有的值都是唯一的，则不会按username排序。

指定排序方向

数据排序不限于升序排序（从 A 到 Z ）。这只是默认的排序顺序，还可以使用ORDER BY子句以降（从 Z 到 A ）顺序排序。为了进行**降序排序**，必须指定DESC关键字。

select id,price,name from products order by price desc,name：以降序排列price，再对name排序

- DESC关键字只应用到**直接位于其前面**的列名。在上例中，只对price列指定DESC，对name列不指。因此，price列以降序排序，而name列（在每个价格内）仍然按标准的升序排序。

在多个列上降序排序

- 如果想在多个列上进行降序排序，必须对每个列指定 DESC关键字。

与DESC相反的关键字是ASC（ASCENDING），在升序排序时可以指定它。但实际上，ASC没有多用处，因为升序是默认的（如果既不指定ASC也不指定DESC，则假定为ASC）。

- 使用 ORDER BY和LIMIT的组合，能够找出一个列中最高或最低的值

select price from products order by price desc limit 1;

price DESC 保证行是按照price最大到最小检索的，而LIMIT 1告诉MySQL仅返回一行。

过滤数据

使用WHERE子句

数据库表一般包含大量的数据，很少需要检索表中所有行。通常只会根据特定操作或报告的需要提取数据的子集。只检索所需数据需要指定搜索条件（search criteria），搜索条件也称为过滤条件（filter ondition）。

● 在 SELECT语句中，数据根据WHERE子句中指定的搜索条件进行过滤。WHERE子句在表名（FRM子句）之后给出

select price from products where price=2;

● WHERE子句的位置

在同时使用ORDER BY和WHERE子句时，应该让ORDER BY位于WHERE之后，否则将会产生错误

WHERE子句操作符

操作符	说明
=	等于
<>	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

● 何时使用引号

如果仔细观察上述 WHERE 子句中使用的条件，

单引号用来限定字符串。如果将值与串类型的列进行比较，则需要限定引号。用来与数值列进行比较值不用引号。

● BETWEEN操作符的语法

为了检查某个范围的值，可使用BETWEEN操作符。其语法与其他WHERE子句的操作符稍有不同，因它需要两个值，即范围的开始值和结束值。

例如：select price from products where price between 5 and 10;

在使用BETWEEN时，必须指定两个值——所需范围的低端值和高端值。这两个值必须用AND关键字隔。BETWEEN匹配范围中所有的值，包括指定的开始值和结束值。

空值检查

在创建表时，表设计人员可以指定其中的列是否可以不包含值。在一个列不包含值时，称其为包含空值

ULL。

- NULL：无值（no value），它与字段包含0、空字符串或仅仅包含空格不同。
- SELECT语句有一个特殊的WHERE子句，可用来检查具有NULL值的列。这个WHERE子句就是IS NULL子句。其语法如下：

select price from products where price IS NULL;

- NULL 与不匹配

在通过过滤选择出不具有特定值的行时，你可能希望返回具有NULL值的行。但是，不行。因为未知有特殊的含义，数据库不知道它们是否匹配，所以在匹配过滤或不匹配过滤时不返回它们。因此，在滤数据时，一定要验证返回数据中确实给出了被过滤列具有NULL的行。

数据过滤

组合WHERE子句

为了进行更强的过滤控制，MySQL允许给出多个WHERE子句。这些子句可以两种方式使用：以AND句的方式或OR子句的方式使用。

操作符（operator） 用来联结或改变WHERE子句中的子句的关键字。也称为逻辑操作符（logical operator）。

AND操作符

为了通过不止一个列进行过滤，可使用AND操作符给WHERE子句附加条件。

例如：select id,price,name from products where id=100 and price<=20;

OR操作符

OR操作符与AND操作符不同，它指示MySQL检索匹配任一条件的行。

例如：select name,price from products where id=100 or id=105;

计算次序

WHERE可包含任意数目的AND和OR操作符。允许两者结合以进行复杂和高级的过滤。

- 例如： select name,price from products where id=100 or id=105 and price<=100;

上述命令中由于AND在计算次序中优先级更高，操作符被错误地组合了。此问题的解决方法是使用圆号明确地分组相应的操作符：select name,price from products where (id=100 or id=105) and price<=100;

- 在WHERE子句中使用圆括号

任何时候使用具有AND和OR操作符的WHERE子句，都应该使用圆括号明确地分组操作符。不要过分赖默认计算次序，即使它确实是你想要的东西也是如此。使用圆括号没有什么坏处，它能消除歧义。

IN操作符

圆括号在WHERE子句中还有另外一种用法。IN操作符用来指定条件范围，范围中的每个条件都可以

行匹配。IN取合法值的由逗号分隔的清单，全都括在圆括号中。

例如：select name,price from products where id IN (104,105); order by name;

IN操作符完成与OR相同的功能：select name,price from products where id 104 or 105 order by ame;

● IN操作符的优点

1. 在使用长的合法选项清单时， IN操作符的语法更清楚且更直观。
2. 在使用 IN时，计算的次序更容易管理（因为使用的操作符更少）。
3. IN操作符一般比OR操作符清单执行更快。
4. IN的最大优点是可以包含其他SELECT语句，使得能够更动态地建立WHERE子句。

NOT操作符

WHERE子句中的NOT操作符有且只有一个功能，那就是否定它之后所跟的任何条件。

##用通配符进行过滤

LIKE操作符

通配符（wildcard）：用来匹配值的一部分的特殊字符

搜索模式（search pattern）由字面值、通配符或两者组合构成的搜索条件

通配符本身实际是SQL的WHERE子句中有特殊含义的字符，SQL支持几种通配符。

● 谓词

操作符何时不是操作符？答案是在它作为谓词（predi-cate）时。从技术上说， LIKE 是谓词而不是作符。虽然最终的结果是相同的，但应该对此术语有所了解，以免在SQL文档

中遇到此术语时不知道。

百分号(%)通配符

最常使用的通配符是百分号（％）。在搜索串中， % 表示任何字符出现任意次数。例如，为了找出有以词admin起头的，可使用以下 SELECT语句：

select id,usernmae from products where id like 'admin%';

此例子使用了搜索模式'admin%'。在执行这条子句时，将检索任意admin起头的词。%告诉MySQL接dmin之后的任意字符，不管它有多少字符

● 通配符可在搜索模式中任意位置使用，并且可以使用多个通配符。下面的例子使用两个通配符，它位于模式的两端：

select id,username from products like id='%user%';

● 通配符也可以出现在搜索模式的中间，虽然这样做不太有用。下面的例子找出以 s起头以e结尾的:

select id,username from products where id like 's%e';

● 重要的是要注意到，除了一个或多个字符外， % 还能匹配0个字符。 %代表搜索模式中给定位置的个、1个或多个字符。

● 注意尾空格

尾空格可能会干扰通配符匹配。例如，在保存词admin 时，如果它后面有一个或多个空格，则子句WERE name LIKE '%admin' 将不会匹配它们，因为在最后的'n'后有多余的字符。解决这个问题的一个单的办法是在搜索模式最后附加一个 % 。一个更好的办法是使用函数去掉首尾空格。

- 注意NULL

虽然似乎%通配符可以匹配任何东西，但有一个例外，即NULL。即使是WHERE name LIKE '%' 也不匹配用值NULL作为产品名的行。

下划线（ _ ）通配符

下划线的用途与%一样，但下划线只匹配单个字符而不是多个字符。

与%能匹配0个字符不一样，_总是匹配一个字符，不能多也不能少

- 使用通配符的技巧

正如所见，MySQL的通配符很有用。但这种功能是有代价的：通配符搜索的处理一般要比前面讨论其他搜索所花时间更长。

1. 不要过度使用通配符。如果其他操作符能达到相同的目的，应该使用其他操作符。
2. 在确实需要使用通配符时，除非绝对有必要，否则不要把它们用在搜索模式的开始处。把通配符置搜索模式的开始处，搜索起来是最慢的。
3. 仔细注意通配符的位置。如果放错地方，可能不会返回想要的数据。

使用正则表达式搜索

用匹配、比较和通配操作符寻找数据。对于基本的过滤（或者甚至是某些不那么基本的过滤），这样足够了。但随着过滤条件的复杂性的增加，WHERE子句本身的复杂性也有必要增加。

- 正则表达式是用来匹配文本的特殊的串（字符集合）。

MYSQL正则表达式

仅为正则表达式语言的一个子集 如果你熟悉正则表达式，需要注意：MySQL仅支持多数正则表达式现的一个很小的子集。

基本字符匹配

- 例:下面的语句检索列　name包含文本admin的所有行：

select name from products where name regexp 'admin';

除关键字LIKE被 EGEXP替代外，这条语句看上去非常像使用LIKE的语句;它告诉MySQL：REGEXP后跟的东西作为正则表达式（与文字正文admin匹配的一个正则表达式）处理。

- 例:　select name from products where name regexp '.000';这里使用了正则表达式 .000。
- .是正则表达式语言中一个特殊的字符。它表示匹配任意**一个字符**，因此，1000和2000都匹配且返。
- LIKE 与 REGEXP区别:

LIKE 匹配整个列。如果被匹配的文本在列值中出现， LIKE 将不会找到它，相应的行也不被返回（除

使用

通配符）。而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，应的行将被返回。这是一个非常重要的差别。

- 如果用REGEXP来匹配整个列值,使用 ^ 和 $ 定位符（anchor）即可
- 匹配不区分大小写

MySQL中的正则表达式匹配（自版本3.23.4后）不区分大小写（即，大写和小写都匹配）。为区分大写，可使用BINARY关键字，如 WHERE prod_name REGEXP BINARY 'Admin .000' 。

进行OR匹配

为搜索两个串之一（或者为这个串，或者为另一个串），使用 |

- 例: select name from products where name regexp '1000|2000'

语句中使用了正则表达式 1000|2000;| 为正则表达式的OR操作符。它表示匹配其中之一，因此 1000 和2000 都匹配并返回。

- 两个以上的 OR 条件

可以给出两个以上的OR条件。例如，'1000 | 2000 | 3000' 将匹配 1000 或 2000 或 3000 。

匹配几个字符之一

匹配任何单一字符,可通过指定一组用 [和] 括起来的字符来完成

- 例: select name from products where name regexp [123] admin;

这里，使用了正则表达式 [123] admin 。 [123] 定义一组字符，它的意思是匹配 1 或 2 或 3 ，因此，1 admin 和 2 admin 都匹配且返回（没有 3 admin ）。

- [] 是另一种形式的 OR 语句。事实上，正则表达式 [123]admin为 [1|2|3]admin 的缩写，也可以使后者。

例:select name from products regexp '1|2|3 admin order by name';

```
+-----------+
| name      |
+-----------+
| 1 admin   |
| 2 admin   |
| user 1000 |
| user 2000 |
+-----------+
```

这并不是期望的输出。两个要求的行被检索出来，但还检索出了另外3行。之所以这样是由于MySQL定你的意思是 '1' 或'2' 或 '3 admin' 。除非把字符 | 括在一个集合中，否则它将应用于整个串

- 字符集合也可以被否定，即，它们将匹配除指定字符外的任何东西。为否定一个字符集，在集合的始处放置一个 ^ 即可。例:[^123]匹配除这些字符外的任何东西。

匹配范围

集合可用来定义要匹配的一个或多个字符。例如[0123456789]

[0-9]等合同[0123456789];范围不限于完整的集合， [1-3] 和 [6-9] 也是合法的范围。此外，范围不定只是数值的，[a-z] 匹配任意字母字符。

例:select name from products where name regexp '[1-5] admin' oeder by name;

匹配特殊字符

正则表达式语言由具有特定含义的特殊字符构成。我们已经看到.、[]、|和-等，还有其他一些字符。

- 为了匹配特殊字符，必须用 \\为前导。\\-表示查找-，\\.表示查找 .
- \\.匹配.，这种处理就是所谓的转义（escaping），正则表达式内具有特殊意义的所有字符都必须这种方式转义。

\\也用来引用元字符（具有特殊含义的字符）

元字符	说明
\f	换页
\n	换行
\r	回车
\t	制表
\v	纵向制表

- 匹配 ''

为了匹配反斜杠（ \ ）字符本身，需要使用 \\\

- \ 或 \?

多数正则表达式实现使用单个反斜杠转义特殊字符，以便能使用这些字符本身。但MySQL要求两个斜杠（MySQL自己解释一个，正则表达式库解释另一个）

匹配字符类

预定义的字符集，称为字符类（character class）

表	说明
[:alnum:]	任意字母和数字（同[a-zA-Z0-9]）
[:alpha:]	任意字符（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:]中的任意符
[:space:]	包括空格在内的任意空白字符（同[\f\n\r\t\

])

[:upper:]	任意大写字母（同[A-Z]）
[:xdigit:]	任意十六进制数字（同[a-fA-F0-9]）

匹配多个实例

重复元字符

元字符	说 明
*	0个或多个匹配
+	1个或多个匹配（等于{1,}）
?	0个或1个匹配（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）

- 例: select name from products where name regexp '\\([0-9] sticks?\\)';

```
+----------------+
| name           |
+----------------+
| TNT (1 stick)  |
| TNT (5 sticks) |
+----------------+
```

正则表达式\\([0-9] sticks?\\):

\\(匹配(，[0-9]匹配任意数字（这个例子中为1和5），sticks?匹配stick和sticks（s后的?使s可选，为?匹配它前面的任何字符的0次或1次出现），\\) 匹配) 。没有？，匹配 stick 和 sticks 会非常困难。

- 匹配连接在一起的4位数字:

select name from products where name regexp '[[:digit:]]{4}' order by name;

等同于:select name from products where name regexp '[0-9][0-9][0-9][0-9]' order by name

定位符

为了匹配特定位置的文本，需要使用定位符

元 字 符	说 明
^	文本的开始
$	文本的结尾
[[:<:]]	词的开始
[[:>:]]	词的结尾

- 符号 ^的用法:**只要是'^'这个字符是在中括号'[]'中被使用的话就是表示字符类的否定;如果不是的就是表示限定开头**

- 使 REGEXP 起类似 LIKE 的作用

本章前面说过，LIKE和REGEXP的不同在于，LIKE匹配整个串而REGEXP匹配子串。利用定位符，通用^开始每个表达式，用$结束每个表达式，可以使REGEXP的作用与LIKE一样

创建计算字段

计算字段

存储在数据库表中的数据一般不是应用程序所需要的格式。我们需要直接从数据库中检索出转换、计或格式化过的数据；而不是检索出数据，然后再在客户机应用程序或报告程序中重新格式化。

- 计算字段并不实际存在于数据库表中。计算字段是运行时在 SELECT语句内创建的
- 字段（field）

基本上与列（column）的意思相同，经常互换使用，不过数据库列一般称为列，而术语字段通常用计算字段的连接上

- 只有数据库知道 SELECT语句中哪些列是实际的表列，哪些列是计算字段。从客户机（如应用程序的角度来看，计算

字段的数据是以与其他列的数据相同的方式返回的。

拼接字段

- 假设有一个 vendors表包含供应商名和位置信息。假如要生成一个供应商报表，需要在供应商的字中按照name(location)这样的格式列出供应商的位置。

  - 此报表需要单个值，而表中数据存储在两个列 vend_name和vend_country中。此外，需要用号将vend_country括起来，这些东西都没有明确存储在数据库表中。

拼接（concatenate） 将值联结到一起构成单个值。

解决办法是把两个列拼接起来。在MySQL的SELECT语句中，可使用Concat()函数来拼接两个列。

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

- Concat() 拼接串，即把多个串连接起来形成一个较长的串。Concat() 需要一个或多个指定的串，个串之间用逗号分隔。

上面的 SELECT 语句连接以下4个元素：

1. 存储在 vend_name 列中的名字；
2. 包含一个空格和一个左圆括号的串；
3. 存储在 vend_country 列中的国家；
4. 包含一个右圆括号的串。

MySQL的不同之处: 多数DBMS使用 + 或 || 来实现拼接，MySQL则使用 Concat() 函数来实现。当把QL语句转换成MySQL语句时一定要注意

- 通过删除数据右侧多余的空格来整理数据，这可以使用MySQL的 RTrim()函数来完成;删除数据右多余的空格:LTrim();去掉串左右两边的空格:Trim()

select concat(rtrim(vend_name),'(',rtrim(vend_country),')');

使用别名

别名（alias）是一个字段或值的替换名。别名用AS关键字赋予

select concat(vend_name,'(',vend_country,')') from as vend_title from vendors order by vend_nme

```
+----------------+
| vend_title     |
+----------------+
| admin1 (USA )  |
| admin2 (JPAN)  |
| admin3 (Eurpoe)|
+----------------+
```

这里的语句中计算字段之后跟了文本AS vend_title。它指示SQL创建一个包含指定计算的名为vend_title的计算字段。从输出中可以看到，结果与以前的相同，但现在列名为vend_title，任何客户机应用都以按名引用这个列，就像它是一个实际的表列一样

- 导出列

别名有时也称为导出列（derived column），不管称为什么，它们所代表的都是相同的东西

执行算术计算

计算字段的另一常见用途是对检索出的数据进行算术计算

例如:select id,price,count, count*price AS total_price from items;

结果将显示一个total_price列,为一个计算字段,此计算为count*price

操作符	说明
+	加
-	减
*	乘
/	除

使用数据处理函数

与其他大多数计算机语言一样，SQL支持利用函数来处理数据。函数一般是在数据上执行的，它给数的转换和处理提供了方便。

使用函数

大多数SQL实现支持以下类型的函数

1. 用于处理文本串（如删除或填充值，转换值为大写或小写）的文本函数。
2. 用于在数值数据上进行算术操作（如返回绝对值，进行代数运算）的数值函数。
3. 用于处理日期和时间值并从这些值中提取特定成分（例如，返回两个日期之差，检查日期有效性等的日期和时间函数。

4. 返回DBMS正使用的特殊信息（如返回用户登录信息，检查版本细节）的系统函数。

文本处理函数

函 数	说 明
Left()	返回串左边的字符
Length()	返回串的长度
Locate()	找出串的一个子串
Lower()	将串转换为小写
LTrim()	去掉串左边的空格
Right()	返回串右边的字符
RTrim()	去掉串右边的空格
Soundex()	返回串的SOUNDEX值
SubString()	返回子串的字符
Upper()	将串转换为大写

- SOUNDEX()是一个将任何文本串转换为描述其语音表示的字母数字模式的算法

日期和时间处理函数

日期和时间采用相应的数据类型和特殊的格式存储，以便能快速和有效地排序或过滤，并且节省物理储空间。

常用日期和时间处理函数

函 数	说 明
AddDate()	增加一个日期（天、周等）
AddTime()	增加一个时间（时、分等）
CurDate()	返回当前日期
CurTime()	返回当前时间
Date()	返回日期时间的日期部分
DateDiff()	计算两个日期之差
Date_Add()	高度灵活的日期运算函数
Date_Format()	返回一个格式化的日期或时间串
Day()	返回一个日期的天数部分
DayOfWeek()	对于一个日期，返回对应的星期几
Hour()	返回一个时间的小时部分
Minute()	返回一个时间的分钟部分
Month()	返回一个日期的月份部分
Now()	返回当前日期和时间

Second()	返回一个时间的秒部分
Time()	返回一个日期时间的时间部分
Year()	返回一个日期的年份部分

● MYSQL使用的日期格式：无论你什么时候指定一日期，不管是插入或更新表值还是用 WHERE子进行过滤，日期必须为格式yyyy-mm-dd。2019年6月18日：2019-06-18;这是首选的日期格式，它除了多义性

1. 例： select id,num from orders where data='2019-06-18';

这会检索出匹配where语句的记录，但是这种写法并不可靠，因为data数据类型为datatime，这种类存储日期以及时间值，表中所有值都具有时间值xx:xx:xx格式，如果用当天时间存储在data中，data=019-06-18 23:11:27，则where子句where data='2019-06-18会匹配失败，将不会检索出记录。

所以要使用Data()函数：Data(data)指示MYSQL仅提取列的日期部分，语句更改为：select id,num from orders where Data(data)='2019-06-18';

如果你想要的仅是日期,则使用 Date()

2. 例： select id,num from orders where Year(data)=2019 and Month(data)=6;

Year()是一个从日期（或日期时间）中返回年份的函数，Month()从日期中返回月份。where子句检出data为2019年6月的所有行

数值处理函数

数值处理函数仅处理数值数据。这些函数一般主要用于代数、三角或几何运算

常用数值处理函数

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()	返回除操作的余数
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

聚集函数

将数据汇总而不是把它们实际检索出来

● 聚集函数（aggregate function）

运行在行组上，计算和返回单个值的函数。

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()	返回某列值之和

分组数据

分组是在SELECT语句的GROUP BY子句中建立的。

例：select vend_id,COUNT(*) AS num_prods from products GROUP BY vend_id;

```
+---------+------------+
| vend_id |  num_prods |
+---------+------------+
| 00001   |          3 |
| 00002   |          5 |
| 00003   |          2 |
+---------+------------+
```

上面的SELECT语句指定了两个列，vend_id包含产品供应商的ID，num_prods 为计算字段（用COUT(*) 函数建立）。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而是整个表

计算num_prods一次。从输出中可以看到vend_id 中0001 prod为3,0002 prod为5，0003 prod为2

使用 GROUP BY 子句的规定

GROUP BY 子句可以包含任意数目的列。这使得能对分组进行嵌套，

为数据分组提供更细致的控制。

- 如果在 GROUP BY 子句中嵌套了分组，数据将在最后规定的分组上进行汇总。换句话说，在建立组时，指定的所有列都一起计算（所以不能从个别的列取回数据）。
- GROUP BY 子句中列出的每个列都必须是检索列或有效的表达式（但不能是聚集函数）。如果在 SEECT 中使用表达式，则必须在GROUP BY 子句中指定相同的表达式。不能使用别名。
- 除聚集计算语句外， SELECT 语句中的每个列都必须在 GROUP BY 子句中给出。
- 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，它们将为一组。
- GROUP BY 子句必须出现在 WHERE 子句之后， ORDER BY 子句之前。

过滤分组

- where过滤行，HAVING过滤分组
- HAVING支持所有WHERE操作符

HAVING和WHERE区别

WHERE 在数据分组前进行过滤，HAVING 在数据分组后进行过滤。

可以在一条语句中同时使用WHERE和HAVING子句

分组和排序

ORDER BY 和 GROUP BY区别

ORDER BY	GROUP BY
排序产生的输出	分组行。但输出可能不是分组的序
任意列都可以使用（甚至非选择的列也可以使用）	可能使用选择列或表达式列，而且必须使用每个选择列表达式
不一定需要	如果与聚集函数一起使用列（或表达），则必须使用

- 不要忘记 ORDER BY

般在使用 GROUP BY 子句时，应该也给出 ORDER BY 子句。这是保证数据正确排序的唯一方法。千不要仅依赖 GROUP BY 排序数据。

SELECT字句顺序

子句	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	仅在从表选择数据时使用
WHERE	行级过滤	否
GROUP BY	分组说明	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	否

使用子查询

SELECT语句是SQL的查询，即从单个数据库表中检索数据的单条语句。

查询（query）:任何SQL语句都是查询。但此术语一般指 SELECT语句

SQL还允许创建子查询（subquery），即嵌套在其他查询中的查询。

利用子查询进行过滤

两条SQL语句：

select num from table1 where id = admin;

select id from table2 where num IN (10,20);

组合(利用子查询)：

select id from table2 where num IN (select num from table1 where id ='admin';)

● 在 SELECT 语句中，子查询总是从内向外处理。在处理上面的SELECT 语句时

相关子查询（correlated subquery）:涉及外部查询的子查询

例：select cust_name,cust_state,(select COUNT(*) from orders where orders.cust_id=customer .cust_id) AS orders from customers ORDER BY cust_name;

这条SELECT语句对customers表中返 回 3 列 ： cust_name 、 cust_state 和 orders 。 orders 是一计算字段，它是由圆括号中的子查询建立的。该子查询对检索出的每个客户执行一次。在此例子中，子查询执行了5次，因为检索出了5个记录。

子查询中的WHERE子句使用了完全限定列名

相关子查询（correlated subquery）:涉及外部查询的子查询

这种类型的子查询称为相关子查询。任何时候只要列名可能有多义性，就必须使用这种语法（表名和名由一个句点分隔）

联结表

SQL最强大的功能之一就是能在数据检索查询的执行中联结（join）表

外键（foreign key）:外键为某个表中的一列，它包含另一个表的主键值，定义了两个表之间的关系

为什么要使用联结

分解数据为多个表能更有效地存储，更方便地处理，并且具有更大的可伸缩性

创建联结

例：select vend_name,prod_name,prod_price from vendors,products where vendors.vend_id = products.vend_id order by vend_name,prod_name;

● SELECT语句与前面所有语句一样指定要检索的列。这里，最大的差别是所指定的两个列（ prod_name和

prod_price ）在一个表中，而另一个列（ vend_name ）在另一个表中。

● 现在来看 FROM 子句。与以前的 SELECT 语句不一样，这条语句的 FROM子句列出了两个表，分是 vendors 和 products。它们就是这条 SELECT语句联结的两个表的名字。这两个表用WHERE子句确联结，WHERE子句指示MySQL匹配vendors表中的vend_id和products表中的vend_id。

笛卡儿积（cartesian product） 由没有联结条件的表关系返回的结果为笛卡儿积。检索出的行的数将是第一个表中的行数乘以第二个表中的行数。

内部联结

之前用的联结称为**等值联结**（equijoin），它基于两个表之间的相等测试。这种联结也称为内部联结

例：select vend_name,prod_name,prod_price from vendors INNER JOIN products ON vendors vend_id = products.vend_id;

这个select语句等同于select vend_name,prod_name,prod_price from vendors,products where vendors.vend_id = products.vend_id order by vend_name,prod_name;

但是from语句不同,两个表都是from子句的组成，以INNER JOIN指定, **这时联结条件用ON而不是WHERE**

传递给 ON 的实际条件与传递给 WHERE 的相同。

ANSI SQL规范首选 INNER JOIN 语法。此外，尽管使用 WHERE 子句定义联结的确比较简单，但是用明确的联结语法能够确保不会忘记联结条件，有时候这样做也能影响性能

联结多个表

SQL对一条 SELECT 语句中可以联结的表的数目没有限制。创建联结的基本规则也相同。首先列出所表，然后定义表之间的关系。例如：

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_nm = 2;

创建高级联结

使用表别名:表别名不仅能用于 WHERE 子句，它还可以用于 SELECT 的列表、 ORDER BY 子句以及句的其他部分。

表别名只在查询执行中使用。与列别名不一样，表别名不返回到客户机。

select cust_name,cust_contact from customers AS c,orders AS o,orderitems AS oi WHERE c.cus_id = o.cust_id AND oi.order_num = o.order_num AND prod_id = 'admin';

自联结

使用表别名的主要原因之一是能在单条 SELECT 语句中不止一次引用相同的表。

● 你发现某物品（其ID为 DTNTR ）存在问题，因此想知道生产该物品的供应商生产的其他物品是否存在这些问题。此查询要求首先找到生产ID为 DTNTR 的物品的供应商，然后找出这个供应商生产的他物品。

```
select prod_id,prod_name from products where vend_id = (
    select vend_id from products where prod_id = 'DTNR';
)
```

这个SQL语句使用了子查询，内部的SELECT语句做了一个简单的检索，返回生产ID为DTNTR的物品应商的vend_id。该ID用于外部查询的WHERE子句中，以便检索出这个供应商生产的所有物品

使用联结的相同查询：

select p1.prod_id,p1.prod_name from products AS p1,products AS p2 where p1.vend_id = p2.end_id AND p2.prod_id = 'DTNR'

此查询中需要的两个表实际上是相同的表，因此products表在FROM子句中出现了两次。虽然这是完合法的，但对products的引用具有二义性，因为MySQL不知道你引用的是products表中的哪个实例为解决此问题，使用了表别名。products的第一次出现为别名 p1 ，第二次出现为别名 p2 。现在可将这些别名用作表名。例如，SELECT语句使用 p1 前缀明确地给出所需列的全名。如果不这样，MySL将返回错误，因为分别存在两个名为prod_id、prod_name的列。MySQL不知道想要的是哪一个列即使它们事实上是同一个列）。WHERE（通过匹配 p1 中的vend_id和 p2 中的vend_id）首先联结

个表，然后按第二个表中的 prod_id 过滤数据，返回所需的数据。

用自联结而不用子查询: 自联结通常作为外部语句用来替代从相同表中检索数据时使用的子查询语句

虽然最终的结果是相同的，但有时候处理联结远比处理子查询快得多。应该试一下两种方法，以确定一种的性能更好。

自然联结

标准的联结返回所有数据，甚至相同的列多次出现。自然联结排除多次出现，使每个列只返回一次。

select c.* o.order_num,o.order_data,oi.prod_id,oi.quantity,oi.item_price from customers as c,oders as o,orderitems as oi where c.cust_id = 0.cust_id and oi.order_num = o.order_num and pod_id = 'FB';通配符只对第一个表使用。所有其他列明确列出，所以没有重复的列被检索出来

外部联结

许多联结将一个表中的行与另一个表中的行相关联。但有时候会需要包含没有关联行的那些行

下面的 SELECT 语句给出一个简单的内部联结。它检索所有客户及其订单：

select customers.cust_id,orders.order_num from customers INNER JOIN orders ON customers cust_id = order.cust_id;

外部联结语法类似。为了检索所有客户，包括那些没有订单的客户，(在一个表中添加一行，而在另一表中没有相对应的记录，如果使用内部联结，将不会返回这条新添加的记录)例如：

select customers.cust_id,orders.order_num from customers LEFT OUTER JOIN orders ON customers.cust_id = orders.cust_id;

这条SELECT语句使用了关键字OUTER JOIN来指定联结的类型（而不是在 WHERE 子句中指定）。

与内部联结关联两个表中的行不同的是，外部联结还包括没有关联行的行在使用OUTER JOIN语法时必须使用RIGHT或LEFT关键字指定包括其所有行的表（ RIGHT 指出的是OUTER JOIN右边的表，而LET指出的是OUTER JOIN左边的表，它以左边的表（也就是customers）作为基准，会选择customer中的所有行，如果该行在orders表中不存在相应的记录，就会以null值（必须要支持null值）显示。。上面的例子使用 LEFT OUTER JOIN 从 FROM子句的左边表（ customers 表）中选择所有行。

笛卡儿积（cartesian product）:由没有联结条件的表关系返回的结果为笛卡儿积。检索出的行的数将是第一个表中的行数乘以第二个表中的行数

外部联结的类型

存在两种基本的外部联结形式：左外部联结和右外部联结。它们之间的唯一差别是所关联的表的顺序同。换句话说，左外部联结可通过颠倒 FROM 或 WHERE 子句中表的顺序转换为右外部联结。因此两种类型的外部联结可互换使用，而究竟使用哪一种纯粹是根据方便而定。

- 使用联结和联结条件

  1. 注意所使用的联结类型。一般我们使用内部联结，但使用外部联结也是有效的。
  2. 保证使用正确的联结条件，否则将返回不正确的数据。
  3. 应该总是提供联结条件，否则会得出笛卡儿积。
  4. 在一个联结中可以包含多个表，甚至对于每个联结可以采用不同的联结类型。虽然这样做是合的，一般也很有用，但应该在一起测试它们前，分别测试每个联结。这将使故障排除更为简单

组合查询

多数SQL查询都只包含从一个或多个表中返回数据的单条 SELECT 语句。MySQL也允许执行多个查询多条 SELECT 语句），并将结果作为单个查询结果集返回。这些组合查询通常称为并（union）或复查询

（compound query）。

有两种基本情况，其中需要使用组合查询：

1. 在单个查询中从不同的表返回类似结构的数据；
2. 对单个表执行多个查询，按单个查询返回数据。

创建组合查询

可用UNION操作符来组合数条SQL查询。利用UNION，可给出多条SELECT 语句，将它们的结果组成单个结果集。

使用UNION

单条语句：

select vend_id,prod_id,prod_price from products where prod_price <= 5;

select vend_id,prod_id,prod_price from products where vend_id IN (100,200);

使用UNION:

```
select vend_id,prod_id,prod_price from products where vend_id IN (100,200);
```

UNION指示MySQL执行两条SELECT语句，并把输出组合成单个查询结果集。

- UNION规则

1. UNION 必须由两条或两条以上的 SELECT 语句组成，语句之间用关键字 UNION 分隔（因此如果组合4条 SELECT 语句，将要使用3个UNION 关键字）。
2. UNION 中的每个查询必须包含相同的列、表达式或聚集函数（不过各个列不需要以相同的次序出）。
3. 列数据类型必须兼容：类型不必完全相同，但必须是DBMS可以隐含地转换的类型（例如，不的数值类型或不同的日期类型）。

包含或取消重复的行

UNION 从查询结果集中自动去除了重复的行这是UNION的默认行为，但是如果需要，可以改变它。实上，如果想返回**所有匹配行**，可使用UNION ALL而不是UNION。

UNION 与 WHERE: UNION 几乎总是完成与多个WHERE 条件相同的工作。 UNION ALL 为 UNION 的一种形式，它完成WHERE 子句完成不了的工作。如果确实需要每个条件的匹配行全部出现（包括复行），则必须使用 UNION ALL 而不是 WHERE 。

对组合查询结果排序

在用 UNION 组合查询时，只能使用一条 ORDER BY 子句，它必须出现在最后一条 SELECT 语句之。

对于结果集，不存在用一种方式排序一部分，而又用另一种方式排序另一部分的情况，因此不允许使多条 ORDER BY 子句。

- 虽然 ORDER BY 子句似乎只是最后一条 SELECT 语句的组成部分，但实际上MySQL将用它来排序有 SELECT 语句返回的所有结果。
- 使用 UNION 的组合查询可以应用不同的表。

全文本搜索

MySQL支持几种基本的数据库引擎。并非所有的引擎都支持本书所描述的全文本搜索。两个最常使的引擎为 MyISAM 和 InnoDB ，前者支持全文本搜索，而后者不支持。

- LIKE 关键字，它利用通配操作符匹配文本（和部分文本）。使用 LIKE ，能够查找包含特殊值或部值的行（不管这些值位于列内什么位置）。

启用全文本搜索支持

一般在创建表时启用全文本搜索。CREATE TABLE语句接受FULLTEXT子句，它给出被索引列的一个号分隔的列表。

例:

```
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;
```

这条CREATE TABLE语句定义表productnotes并列出它所包含的列。这些列中有一个名为note_text列，为了进行全文本搜索，MySQL根据子句FULLTEXT(note_text)的指示对它进行**索引**。这里的FULLTEXT索引单个列，如果需要也可以指定多个列。

在定义之后，MySQL自动维护该索引。在增加、更新或删除行时，索引随之自动更新。

可以在创建表时指定 FULLTEXT ，也可以在稍后指定（**在这种情况下所有已有数据必须立即索引**）。

使用全文本搜索

在索引之后， SELECT 可与 Match() 和 Against() 一起使用以实际执行搜索，其中 Match() 指定被索的列，Against() 指定要使用的搜索表达式

例：select note_text from productnotes where Match(note_text) Against('admin');

- 此 SELECT 语句检索单个列 note_text 。由于WHERE子句，这个全文本搜索被执行。 Match(note_text) 指示MySQL针对指定的列进行搜索， Against('admin') 指定词admin作为搜索文本。由于有n包含词 admin ，这n个行被返回。

使用完整的 Match(): 传递给 Match() 的值必须与FULLTEXT() 定义中的相同。如果指定多个列，则须列出它们（而且次序正确）。

搜索不区分大小写 除非使用BINARY方式，否则全文本搜索不区分大小写。

上面语句等同于select note_text from productnotes where note_text like '%admin%'

● select note_text Match(note_text) Against('admin') AS rank from productnotes这条语句会所有行返回， Match() 和Against()用来建立一个计算列（别名为 rank ），此列包含全文本搜索计算的等级值。等级由MySQL根据行中词的数目、唯一词的数目、整个索引中词的总数以及包含该词的的数目计算出来。

查询扩展

在使用查询扩展时，MySQL对数据和索引进行两遍扫描来完成搜索：

1. 首先，进行一个基本的全文本搜索，找出与搜索条件匹配的所有行；
2. 其次，MySQL检查这些匹配行并选择所有有用的词（我们将会简要地解释MySQL如何断定什么有，什么无用）。
3. 再其次，MySQL再次进行全文本搜索，这次不仅使用原来的条件，而且还使用所有有用的词。

利用查询扩展，能找出可能相关的结果，即使它们并不精确包含所查找的词。、

简单的全文本搜索：select note_text from productnotes where match(note_text) against('admin' ;

使用扩展查询：select note_text from productnotes where match(note_text) against('admin' WI H QUERY EXPANSION);

布尔文本搜索

以布尔方式进行搜索：

1. 要匹配的词；
2. 要排斥的词（如果某行包含这个词，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. 表达式分组；
5. 另外一些内容。

即使没有 FULLTEXT 索引也可以使用

例：select note_text from products where match(note_text) against('admin' IN BOOLEAN MO E)

此全文本搜索检索包含词 heavy 的所有行其中使用了关键字 IN BOOLEAN MODE ，但实际上没有定布尔操作符，

因此，其结果与没有指定布尔方式的结果相同。

为了二匹配'admin'而不包含任意以'zh'开头的词的行，查询语句为：select note_text from products where match(note_text) against('admin -rope*' IN BOOLEAN MODE

-排除一个词，而*是截断操作符（可理解为用于词尾的一个通配符）

全文本布尔操作符

布尔操作符	说 明
+	包含，词必须存在
-	排除，词必须不出现

|包含，而且增加等级值

<|包含，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包含或排除这个短语）

- select note_text from productnotes where match(note_text) against('+admin +zh' IN BOOLAN MODE);这个搜索匹配包含词admin和zh的行。
- select note_text from productnotes where match(note_text) against('admin zh' IN BOOLEAN MODE);没有指定操作符，这个搜索匹配包含admin和zh中的至少一个词的行。
- select note_text from productnotes where match(note_text) against("admin zh" IN BOOLEA MODE);这个搜索匹配短语admin zh而不是匹配两个词admin和zh。
- select note_text from productnotes where match(note_text) against('>admin <root' IN BOOEAN MODE);匹配 admin 和 root，增加前者的等级，降低后者的等级。
- select note_text from productnotes where match(note_text) against('+admin (<root)' IN BOLEAN MODE);这个搜索匹配词 admin 和 root ，降低后者的等级

Q&A

1. 在索引全文本数据时，短词被忽略且从索引中排除。短词定义为那些具有3个或3个以下字符的词（果需要，这个数目可以更改）。
2. MySQL带有一个内建的非用词（stopword）列表，这些词在索引文本数据时总是被忽略。如果需，可以覆盖这个列表
3. 许多词出现的频率很高，搜索它们没有用处（返回太多的结果）。因此，MySQL规定了一条50%则，如果一个词出现在50%以上的行中，则将它作为一个非用词忽略。50%规则不用于IN BOOLEAN MODE。
4. 如果表中的行数少于3行，则全文本搜索不返回结果（因为每个词或者不出现，或者至少出现在50的行中）。
5. 忽略词中的单引号。例如， don't 索引为 dont
6. 不具有词分隔符（包括日语和汉语）的语言不能恰当地返回全文本搜索结果。
7. 仅在 MyISAM 数据库引擎中支持全文本搜索

插入数据

INSERT是用来插入（或添加）行到数据库表的。插入可以用几种方式使用：

1. 插入完整的行；
2. 插入行的一部分；
3. 插入多行；

4. 插入某些查询的结果。

INSERT 语句一般不会产生输出。

插入完整的行

指定表名和被插入到新行中的值:

```
insert into productnotes values(
    'Admin','GanSu','LanZhou','CN','730070','133****1212','example@gmail.com'
);
```

这种语法很简单，但并不安全，应该尽量避免使用。上面的SQL语句高度依赖于**表中列的定义次序**，使可得到这种次序信息，也不能保证下一次表结构变动后各个列保持完全相同的次序。依赖于特定列序的SQL语句是很不安全的。

更安全的INSERT语句：

```
insert into productnotes(
    name,address,city,country,zip,contact,email
) values(
    'Admin','GanSu','LanZhou','CN','730070','133****1212','example@gmail.com'
);
```

这条INSERT语句在表名后的括号里明确地给出了列名。在插入行时，MySQL将用VALUES列表中的应值填入列表中的对应项

- 因为提供了列名， VALUES必须以其指定的次序匹配指定的列名，**不一定按各个列出现在实际表的次序。其优点是，即使表的结构改变，此INSERT语句仍然能正确工作。**
- 不管使用哪种 INSERT语法，都必须给出VALUES的正确数目。如果不提供列名，则必须给每个表提供一个

值。如果提供列名，则必须对每个列出的列给出一个值。如果不这样，将产生一条错误消息，相应的插入不成功。

- 使用这种语法，还可以省略列。这表示可以只给某些列提供值，给其他列不提供值
- 满足在INSERT操作中 **省略列**的条件

  1. 该列定义为允许NULL值
  2. 该列定义了默认值

插入多个行

可以使用多条INSERT语句，每条语句用分号分开

性能：MySQL用单条 INSERT 语句处理多个插入比使用多条 INSERT语句快。

插入检索出的语句

可以利用SELECT语句的结果将其插入表中，它由一条INSERT语句和一条SELECT语句组成

例：把一个名为old的表中的数据导入new表中，表new和old的结构相同

```
insert into new(
```

```
    name,address,city,country,zip,contact,email
)
select
    name,address,city,country,zip,contact,email
from old;
```

这个例子使用 INSERT SELECT 从old中将所有数据导入new

更新和删除数据

更新

使用UPDATE语句，有两种方式：

1. 更新表中特定行
2. 更新表中所有行

基本的UPDATE语句由三个部分组成：

3. 要更新的表；
4. 列名和它们的新值；
5. 确定要更新行的过滤条件。

例：`UPDATE productnotes SET email='0x01@qq.com' WHERE id = '001'`

UPDATE 语句总是以需要更新的表的名字开始，SET子句设置email列指定的值，WHERE子句告诉MYSQL需要更新的行，**如果没有WHERE子句，MYSQL会更新表中所有的行，需要注意**

更新多个列：

```
    name = 'admin',
    email = '0x01@qq.com'
WHERE id = '001';
```

在更新多个列时，只需要使用单个 SET 命令，每个“列 = 值”对之间用逗号分隔

IGNORE 关键字: 如果用 UPDATE 语句更新多行，并且在更新这些行中的一行或多行时出一个现错误则整个 UPDATE 操作被取消（错误发生前更新的所有行被恢复到它们原来的值）,可以使用关键字'IGORE'，即使发生错误，也可以继续进行更新

- 为了删除某个列的值，可设置它为 NULL

`UPDATE table1 SET s = NULL where id = 1;`其中 NULL 用来去除s列中的值

删除数据

使用 DELETE 语句。

可以两种方式使用 DELETE ：

1. 从表中删除特定的行；
2. 从表中删除所有行。

如果没有WHERE子句，MYSQL会删除表中所有的行，需要注意

例：DELETE FROM table1 WHERE id = '1';

● DELETE 不需要列名或通配符。 DELETE 删除整行而不是删除列。为了删除指定的列，请使用 UPDTE 语句(设置列值为NULL)。

如果想从表中删除所有行，则不使用 DELETE 。可使用 TRUNCATE TABLE 语句，它完成相同的工作但速度更

快（ TRUNCATE 实际是删除原来的表并重新创建一个表，而不是逐行删除表中的数据）。

UPDATE和DELETE需要遵循的原则

1. 除非确实打算更新和删除每一行，否则绝对不要使用不带 WHERE子句的 UPDATE 或 DELETE 语。
2. 保证每个表都有主键，尽可能像 WHERE 子句那样使用它（可以指定各主键、多个值或值的范围）。
3. 在对 UPDATE 或 DELETE 语句使用 WHERE 子句前，应该先用 SELECT 进行测试，保证它过滤的正确的记录，以防编写的 WHERE 子句不正确。
4. 使用强制实施引用完整性的数据库，这样MySQL将不允许删除具有与其他表相关联的数据的行。

MYSQL没有撤销操作，所以需要小心使用UPDATE和DELETE

创建和操作表

创建表

CREATE TABLE语句，为了使用CREATE TABLE，必须给出以下信息：

1. 新表的名字，在关键字 CREATE TABLE 之后给出；
2. 表列的名字和定义，用逗号分隔。

例：

```
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;
```

● 在创建新表时，指定的表名必须不存在，否则将出错。如果要防止意外覆盖已有的表，SQL要求首手工删除该表，然后再重建它，而不是简单地用创建表语句覆盖它

● 如果你仅想在一个表不存在时创建它，应该在表名后给出 IF NOT EXISTS。这样做不检查已有表模式是否与你打算创建的表模式相匹配。它只是查看表名是否存在，并且仅在表名不存在时创建它。

NULL值

NULL值就是没有值或者缺值。允许 NULL 值的列也允许在插入行时不给出该列的值。不允许 NULL

的列不接受该列没有值的行

也就是说，如果某个列含有关键字NOT　NULL，如果插入没有值的列，将会返回错误，且插入失败

● 不要把 NULL 值与空串相混淆。 NULL 值是没有值，它不是空串。如果指定 ''（两个单引号，其没有字符），这在 NOT NULL 列中是允许的。空串是一个有效的值，它不是无值。 NULL 值用关键字 NULL 而不是空串指定。

主键

主键值必须唯一。

即：表中的每个行必须具有唯一的主键值。如果主键使用单个列，则它的值必须唯一。如果使用多个，则这些列的组合值必须唯一。

● 主键用以下类似的语句定义：　PRIMARY KEY(note_id)

● 为创建由多个列组成的主键，应该以逗号分隔的列表给出各列名：　PRIMARY KEY(note_id,note_ame)

● 主键为其值唯一标识表中每个行的列。主键中只能使用不允许 NULL 值的列。允许 NULL 值的列不作为唯一标识。

使用AUTO_INCREMENT

note_id　int　　NOT NULL AUTO_INCREMENT,

AUTO_INCREMENT 告诉MySQL，本列每当增加一行时自动增量。每次执行一个INSERT操作时，MSQL自动对该列增量（从而才有这个关键字AUTO_INCREMENT），给该列赋予下一个可用的值。样给每个行分配一个唯一的 note_id，从而可以用作主键值。

● 每个表只允许一个 AUTO_INCREMENT 列，而且它必须被索引（如，通过使它成为主键）

插入默认值

如果在插入行时没有给出值，MySQL允许指定此时使用的默认值。默认值用CREATE TABLE语句的定义中的 DEFAULT关键字指定。

```
CREATE TABLE table1(
    id      int(20)     NOT NULL,
    name    char(10)    NOT NULL DEFAULT 'tourist'
)
```

MYSQL 不允许使用函数作为默认值，它只支持常量

引擎类型

与其他DBMS一样，MySQL有一个具体管理和处理数据的内部引擎。在你使用 CREATE TABLE 语句，该引擎具体创建表，而在你使用 SELECT语句或进行其他数据库处理时，该引擎在内部处理你的请。多数时候，此引擎都隐藏在DBMS内，不需要过多关注它。

● MySQL具有多种引擎：它打包多个引擎，这些引擎都隐藏在MySQL服务器内，全都能执行 CREATE TABLE 和 SELECT等命令

当忽略这些数据库引擎，省略ENGINE= 语句，则使用默认引擎(很可能是MyISAM)

几个引擎：

1. InnoDB 是一个可靠的事务处理引擎（参见第26章），它不支持全文本搜索；
2. MEMORY 在功能等同于 MyISAM，但由于数据存储在内存（不是磁盘）中，速度很快（特别适于临时表）；
3. MyISAM 是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。

外键不能跨引擎

更新表

使用ALTER TABLE语句

为了使用ALTER TABLE更改表结构，必须给出下面的信息：

1. 在 ALTER TABLE之后给出要更改的表名（该表必须存在，否则将出错）；
2. 所做更改的列表

给表添加一个列，必须明确其数据类型：

ALTER TABLE table1 ADD c1 char(20);

删除刚才添加的列

ALTER TABLE table1 DROP COLUMN c1;

- ALTER TABLE 的一种常见用途是定义外键

复杂的表结构更改一般需要手动删除过程，需要以下几个步骤

1. 用新的列布局创建一个新表；
2. 使用 INSERT SELECT 语句
3. 检验包含所需数据的新表；
4. 重命名旧表（如果确定，可以删除它）；
5. 用旧表原来的名字重命名新表；
6. 根据需要，重新创建触发器、存储过程、索引和外键。

删除表

删除表（删除整个表而不是其内容），使用DROP TABLE语句

重命名表

使用RENAME TABLE语句

RENAME TABLE table1 TO t1;

对多个表重命名

```
RENAME TABLE table1 TO t1,
RENAME TABLE table2 TO t2,
RENAME TABLE table3 TO t3,
RENAME TABLE table4 TO t4;
```

使用视图

视图是虚拟的表，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。

视图的一些常见应用

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的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。

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，必须具有足够的访问权限。这些限制通常由数据库管理人员授予。
4. 视图可以嵌套，即可以利用从其他视图中检索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 中也含有 ORDER BY ，那么该视中的 ORDER BY 将被覆盖。
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。例如，编写一条联结表和视图的SELECT语句。

视图的创建

1. 视图用 CREATE VIEW语句来创建。
2. 使用 SHOW CREATE VIEW viewname;；来查看创建视图的语句。
3. 用 DROP删除视图，其语法为DROP VIEW viewname;。
4. 更新视图时，可以先用DROP再用CREATE，也可以直接用 CREATE ORREPLACE VIEW。如果要新的视图不存在，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 条更新语句会换原有视图。

利用视图简化复杂的联结

利用视图可以隐藏复杂的SQL语句

CREATE VIEW productcustomers AS SELECT cust_name,cust_contact,prod_id FROM customers orders,orderitems WHERE customers.cust_id = orders.cust_id AND orderitems.order_num = o ders.order_num

检索prod_id为'admin'：SELECT cust_name,cust_contact FROM productcustomers where prod_ d = 'admin';这条语句通过 WHERE 子句从视图中检索特定数据。在MySQL处理此查询时，它将指定 WHERE 子句添加到视图查询中的已有WHERE子句中，以便正确过滤数据。

● 视图极大地简化了复杂SQL语句的使用。利用视图，可一次性编写基础的SQL，然后根据需要多次用。

更新视图

对视图使用 INSERT 、 UPDATE 和 DELETE ）。更新一个视图将更新其基表（视图本身没有数据）如果对视图增加或删除行，实际上是对其基表增加或删除行。

如果视图定义中有以下操作，则不能进行视图的更新：

1. 分组（使用 GROUP BY 和 HAVING ）；
2. 联结；
3. 子查询；
4. 并；
5. 聚集函数（ Min() 、 Count() 、 Sum() 等）；
6. DISTINCT；
7. 导出（计算）列。

存储过程

MySQL5添加了对存储过程的支持

存储过程简单来说，就是为以后的使用而保存的一条或多条MySQL语句的集合。可将其视为批文件虽然它们的作用

不仅限于批处理。

存储过程（Stored Procedure）是一种在数据库中存储复杂程序，以便外部程序调用的一种数据库象。

执行存储过程

MySQL称存储过程的执行为调用，因此MySQL执行存储过程的语句为 CALL；CALL 接受存储过程的字以及需要传递给它的任意参数

例：执行名为 productpricing 的存储过程，它计算并返回产品的最低、最高和平均价格。

```
            @pricelow
            @priceheight
            @priceaverage
);
```

创建存储过程

一个返回产品平均价格的存储过程。此存储过程名为productpricing ，用CREATE PROCEDURE productpricing()语句定义。如果存储过程接受参数，它们将在 () 中列举出来，BEGIN 和 END 语句用来定存储过程体，过程体本身仅是一个简单的 SELECT 语句

```
CREATE PROCEDURE productpricing()
BEGIN
   SELECT Avg(prod_price) AS priceaverage FROM products
END;
```

在MySQL处理这段代码时，它创建一个新的存储过程 productpricing 。没有返回数据，因为这段代并未调用存储过程，这里只是为以后使用而创建它。

使用这个存储过程：CALL productpricing();

存储过程实际上是一种函数，所以存储过程名后需要有 () 符号（即使不传递参数也需要）

删除存储过程

语句：DROP PROCEDURE productpricing;

使用参数

变量（variable）：内存中一个特定的位置，用来临时存储数据

例：

```
CREATE PROCEDURE productpricing(
   OUT pl DECIMAL(8,2)
   OUT ph DECIMAL(8,2)
   OUT pa DECIMAL(8,2)
)
BEGIN
   SELECT Min(prod_price) INTO pl FROM products;
   SELECT Max(prod_price) INTO ph FROM products;
   SELECT Avg(prod_price) INTO pa FROM products;
END;
```

1. 此存储过程接受3个参数： pl:pricelow， ph:priceheight， pa:priceaverage
2. 这里使用十进制值（DECIMAL）
3. 关键字 OUT 指出相应的参数用来从存储过程传出一个值（返回给调用者）
4. IN（传递给存储过程）、OUT（从存储过程传出）和 INOUT （对存储过程传入和传出）类型的参。
5. 存储过程的代码位于 BEGIN 和 END 语句内，，它们是一系列SELECT 语句，用来检索值，然后保到相应的变量（通过指定 INTO 关键字）。

为调用此修改过的存储过程，必须指定3个变量名:

```
CALL productpricing(@pricelow
            @priceheight
            @priceaverage);
```

变量名: 所有MySQL变量都必须以 @ 开始

在调用时，这条语句并不显示任何数据。它返回以后可以显示（或在其他处理中使用）的变量

为了显示检索出的产品平均价格:SELECT @priceaverage;

为了获得3个值:SELECT @pricelow,@priceheight,@priceaverage;