

目录

MySQL学习笔记	1.1
-----------	-----

Part I 查询

简单查询 (SELECT 语句)	2.1
排序查询 (ORDER By语句)	2.2
过滤数据 (WHERE子句)	2.3
MySQL库函数	2.4
分组查询 (GROUP BY语句)	2.5
子查询(IN)	2.6
连接查询(JOIN)	2.7
组合查询(UNION)	2.8

Part II 增删改

插入数据(INSERT)	3.1
更新数据(UPDATE)	3.2
删除数据(DELETE)	3.3
增删改表(CREATE、ALTER、DROP)	3.4
MySQL数据类型	3.5

Part III 高级特性

视图(VIEW)	4.1
存储过程(PROCEDURE)	4.2
触发器(TRIGGER)	4.3
事务(TRANSACTION)	4.4
MySQL用户管理	4.5

- [Introduction](#)
- [Summary](#)
 - [Part I 查询](#)
 - [Part II 增删改](#)
 - [Part III 高级特性](#)

Introduction

本文是《MySQL必知必会》，人民邮电出版社，的学习笔记，请购买正版图书。

Summary

- [MySQL学习笔记](#)

Part I 查询

- [简单查询 \(SELECT 语句\)](#)
- [排序查询 \(ORDER By语句\)](#)
- [过滤数据 \(WHERE子句\)](#)
- [MySQL库函数](#)
- [分组查询 \(GROUP BY语句\)](#)
- [子查询\(IN\)](#)
- [连接查询\(JOIN\)](#)
- [组合查询\(UNION\)](#)

Part II 增删改

- [插入数据\(INSERT\)](#)
- [更新数据\(UPDATE\)](#)
- [删除数据\(DELETE\)](#)
- [增删改表\(CREATE、ALTER、DROP\)](#)
- [MySQL数据类型](#)

Part III 高级特性

- [视图\(VIEW\)](#)
- [存储过程\(PROCEDURE\)](#)
- [触发器\(TRIGGER\)](#)
- [事务\(TRANSACTION\)](#)
- [MySQL用户管理](#)

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间: 2020-03-27 00:47:53

- 简单查询语句(SELECT 查询语句)
 - 检索单个列
 - 检索多个列
 - 检索所有列
 - 检索不同的行（去重复值）
 - 限制结果(返回行数)
 - 使用完全限定的表名

简单查询语句(SELECT 查询语句)

检索单个列

```
SELECT 列名 FROM 表名
```

检索多个列

```
SELECT 列名1,列名2 FROM 表名
```

检索所有列

```
SELECT * FROM 表名
```

一般，除非你确实需要表中的每个列，否则最好别使用 * 通配符。虽然使用通配符可能会使你省事，不用明确列出所需列，但检索不需要的列通常会降低检索和应用程序的性能。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列。

检索不同的行（去重复值）

```
SELECT DISTINCT 列名 FROM 表名`
```

不能部分使用**DISTINCT** DISTINCT关键字应用于所有列而不仅是前置它的列。如果给出SELECT DISTINCT 列1, 列2 FROM 表名，除非指定的两个列都不同，否则所有行都将被检索出来。

限制结果(返回行数)

```
SELECT * FROM 表名 LIMIT 开始行数, 返回行数
SELECT * FROM 表名 LIMIT 返回行数 OFFSET 开始行数  仅MySQL 5以上支持
```

例：返回表中前五条数据

```
SELECT * FROM 表名 LIMIT 0,5
或MySQL5+ `SELECT * FROM 表名 LIMIT 5 OFFSET 0
```

返回表中第6~10条数据

```
SELECT * FROM 表名 LIMIT 5,5  
或MySQL5+ SELECT * FROM 表名 LIMIT 5 OFFSET 5
```

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13 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4 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。LIMIT 4 OFFSET 3意为从行3开始取4行，就像LIMIT 3, 4一样。

使用完全限定的表名

```
SELECT 表名.列名 FROM 表名
```

完全限定名字可以在联合查询的时候避免多表的列重名的情况。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间： 2020-01-16 08:00:17

- 排序查询 (ORDER BY语句)
 - 排序数据
 - 按多个列排序
 - 指定排序方向

排序查询 (ORDER BY语句)

排序数据

```
SELECT * FROM 表名 ORDER BY 列名 (要排序的列)
```

通过非选择列进行排序 通常，ORDERBY子句中使用的列将是为显示所选择的列。但是，实际上并不一定要这样，用非检索的列排序数据是完全合法的。

按多个列排序

```
SELECT * FROM 表名 ORDER BY 列名1 (要排序的列), 列名2
```

按照列1先排序，当列1相同时，按照列2排序

指定排序方向

默认是升序 (A~Z字母顺序) 排列

```
SELECT * FROM 表名 ORDER BY 列名  
SELECT * FROM 表名 ORDER BY 列名 ASC
```

降序排列(从Z到A)

```
SELECT * FROM 表名 ORDER BY 列名 DESC
```

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

区分大小写和排序顺序 在对文本性的数据进行排序时，A与a相同吗？a位于B之前还是位于Z之后？这些问题不是理论问题，其答案取决于数据库如何设置。

在字典(dictionary)排序顺序中，A被视为与a相同，这是MySQL (和大多数数据库管理系统)的默认行为。但是，许多数据库 管理员能够在需要时改变这种行为(如果你的数据库包含大量外语字符，可能必须这样做)。

这里，关键的问题是，如果确实需要改变这种排序顺序，用简单的ORDER BY子句做不到。你必须请求数据库管理员的帮助

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间： 2020-01-12 11:36:25

- 过滤数据（WHERE）子句
 - 过滤数据
 - WHERE子句操作符
 - 单值查询
 - 不匹配查询
 - 范围查询
 - 空值查询
 - 并列查询（AND操作符）
 - 任一查询（OR操作符）
 - AND OR 优先级
 - 范围查询（IN操作符）
 - 不匹配查询（NOT IN操作符）
 - 用通配符进行过滤（LIKE操作符）
 - 正则表达式

过滤数据（WHERE）子句

过滤数据

```
SELECT * FROM 表名 WHERE 条件
```

例：

```
SELECT * FROM products WHERE prod_price='14.99'
```

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

例：

```
SELECT * FROM products WHERE prod_price='14.99' ORDER BY prod_price
```

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

单值查询

注意！**MySQL在执行匹配时默认不区分大小写**，如fuses与Fuses匹配。

```
SELECT * FROM 表名 WHERE prod_name='fuses';
```

不匹配查询

```
SELECT * FROM 表名 WHERE 列名 <> '值';
或者 SELECT * FROM 表名 WHERE 列名 != '值';
```

如：输出不是fuses的结果

```
SELECT * FROM products WHERE prod_name <> 'fuses'
或者 SELECT * FROM products WHERE prod_name != 'fuses';
```

何时使用引号 如果仔细观察上述WHERE子句中使用的条件， 会看到有的值括在单引号内(如前面使用的'fuses')而有的值未括起来。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的值不用引号。

范围查询

```
SELECT * FROM 表名 WHERE 列名 BETWEEN 条件A AND 条件B ;
```

例：查询价格在1~11之间的记录

```
SELECT * FROM products WHERE prod_price BETWEEN 1 and 11
```

空值查询

注意：**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 * FROM 表名 WHERE 列名 IS NULL ;
```

并列查询（AND操作符）

```
SELECT * FROM 表名 WHERE 条件1 AND 条件2;
```

例：供应商1003，价格<30的产品信息

```
SELECT * FROM products WHERE prod_price <30 AND vend_id =1003;
```

任一查询（OR操作符）

```
SELECT * FROM 表名 WHERE 条件1 OR 条件2;
```

例：供应商是1003或者价格<30的产品信息

```
SELECT * FROM products WHERE prod_price <30 OR vend_id =1003;
```

AND OR 优先级

SQL(像多数语言一样)在处理OR操作符前，优先处理AND操作符。

在WHERE子句中使用圆括号，消除奇异。

范围查询（IN操作符）

IN操作符用来指定条件范围，范围中的每个条件都可以进行匹配。IN取合法值的由逗号分隔的清单，全都括在圆括号中。

下列两行代码等价

```
SELECT * FROM 表名 WHERE 列名 IN (条件1,条件2);  
SELECT * FROM 表名 WHERE 列名 = 条件1 OR 列名 = 条件2;
```

IN操作符一般比OR操作符清单执行更快。

不匹配查询（NOT IN操作符）

NOT IN操作符在WHERE子句中用来否定后跟条件的关键字。

下列两行代码等价

```
SELECT * FROM 表名 WHERE 列名 NOT IN (条件1,条件2);  
SELECT * FROM 表名 WHERE 列名 != 条件1 AND 列名 != 条件2;
```

例：查询所有供应商不是1002，1003 的产品。


```
SELECT * FROM products WHERE vend_id NOT IN (1002,1003);
SELECT * FROM products WHERE vend_id !=1002 AND vend_id!=1003;
```

MySQL中的NOT MySQL支持使用NOT对IN、BETWEEN和EXISTS子句取反。

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。

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的其他搜索所花时间更长。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。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。把通配符置于搜索模式的开始处，搜索起来是最慢的。

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。

```
SELECT * FROM 表名 WHERE 列名 LIKE '条件%'
```

例如，为了找出所有以词jet起头的顾客姓名，可使用以下SELECT 语句：

```
SELECT * FROM customers WHERE cust_name LIKE 'jet%'
```

jet结尾

```
SELECT * FROM customers WHERE cust_name LIKE '%jet'
```

包含jet

```
SELECT * FROM customers WHERE cust_name LIKE '%jet%'
```

注意：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 prod_name LIKE '%'也不能匹配用值NULL作为产品名的行。只能用下面语句匹配NULL

```
SELECT * FROM 表名 WHERE 列名 IS NULL;
```

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，_总是匹配一个字符，不能多也不能少。

```
SELECT * FROM 表名 WHERE 列名 LIKE '条件%'
```

例：查询供应商ID（4位数）是100开头的，100_

```
SELECT * FROM products WHERE vend_id LIKE '100_';
```

正则表达式

```
SELECT * FROM 表名 WHERE 列名 REGEXP '正则表达式'
```

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间: 2020-01-16 08:00:10

- MySQL库函数
 - 文本处理函数
 - Soundex 函数（按音节查询）
 - Concat函数（拼接字符串）
 - Trim 函数（去除空格）
 - AS操作符（别名）
 - 算术操作符
 - 日期和时间处理函数
 - 数值处理函数
 - 聚集函数
 - 平均数函数（AVG）
 - 计数函数（COUNT）
 - 最大值函数（MAX）
 - 最小值函数（MIN）
 - 求和函数（SUM）
 - 排除重复值

MySQL库函数

注意：函数没有SQL的可移植性强，大多数函数可能是MySQL独有的，或者名称不一样。

函数大多数按照如下分类。

- 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。
- 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。
- 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。
- 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

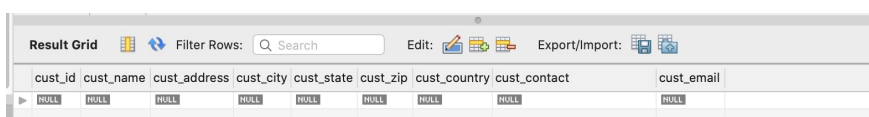
函数	说明
Left()	返回串左边的字符
Length()	返回串的长度
Locate()	找出串的一个子串
Lower()	将串转换为小写
LTrim()	去掉串左边的空格
Right()	返回串右边的字符
RTrim()	去掉串右边的空格
Soundex()	返回串的SOUNDEX值
SubString()	返回子串的字符
Upper()	将串转换为大写

Soundex 函数（按音节查询）

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。SOUNDEX考虑了类似的发音字符和音节，使得能对串进行发音比较而不是字母比较。虽然 SOUNDEX不是SQL概念，但MySQL(就像多数DBMS一样)都提供对SOUNDEX的支持。

下面给出一个使用Soundex()函数的例子。customers表中有一个顾客杰威尔公司，其联系名为周杰伦。但如果这是输入错误，成了周杰轮，怎么办？显然，按正确的联系名搜索不会返回数据，如下所示：

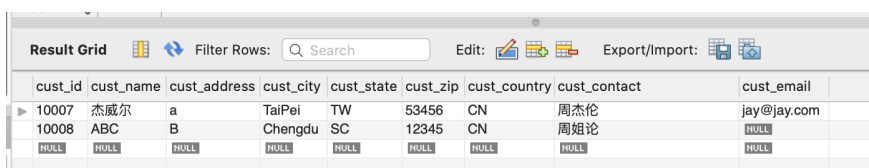
```
SELECT * FROM customers where cust_contact='周杰轮';
```



cust_id	cust_name	cust_address	cust_city	cust_state	cust_zip	cust_country	cust_contact	cust_email
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

现在试一下使用Soundex()函数进行搜索，它匹配所有发音类似于 周杰伦的联系名：

```
SELECT * FROM customers where Soundex(cust_contact)=Soundex('周杰伦');
```



cust_id	cust_name	cust_address	cust_city	cust_state	cust_zip	cust_country	cust_contact	cust_email
10007	杰威尔	a	Taipei	TW	53456	CN	周杰伦	jay@jay.com
10008	ABC	B	Chengdu	SC	12345	CN	周姐伦	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Concat函数（拼接字符串）

拼接(concatenate) 将值联结到一起构成单个值。

```
SELECT CONCAT(字符串1或者列名,'字符串2') FROM 表名;
```

例：把供应商名字和供应商国家连起来

```
SELECT CONCAT(vend_name, '(', vend_country, ')') FROM vendors;
```

Trim 函数（去除空格）

```
SELECT TRIM(列名) FROM 表名;
---- 去除左边空格
SELECT LTRIM(列名) FROM 表名;
---- 去除右边空格
SELECT RTRIM(列名) FROM 表名;
```

AS操作符（别名）

```
SELECT 列名 AS 别名 FROM 表名;
```

算术操作符

操作符	说明
+	加
-	减
*	乘
/	除

```
SELECT 列名1 + 列名2或常量 FROM 表名;
```

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，并且节省物理存储空间。

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

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

聚集函数

我们经常需要汇总数据而不用把它们实际检索出来，为此MySQL提供了专门的函数。使用这些函数，MySQL查询可用于检索数据，以便分析和报表生成。这种类型的检索例子有以下几种。

- 确定表中行数(或者满足某个条件或包含某个特定值的行数)。
- 获得表中行组的和。
- 找出表列(或所有行或某些特定的行)的最大值、最小值和平均值。

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

平均数函数 (AVG)

AVG函数，通过对表中行数计数并计算特定列值之和，求得该列的平均值。**NULL**值 AVG()函数忽略列值为NULL的行。

```
SELECT AVG(列名) FROM 表名;
```

计数函数 (COUNT)

COUNT()函数进行计数。可利用COUNT()确定表中行的数目或符合特定条件的行的数目。

COUNT()函数有两种使用方式。

- 使用COUNT(*)对表中行的数目进行计数，不管表列中包含的是空值(NULL)还是非空值。

返回所有行

```
SELECT COUNT(*) FROM 表名;
```

- 使用COUNT(column)对特定列中具有值的行进行计数，忽略 NULL值。

返回所有非空行

```
SELECT COUNT(列名) FROM 表名;
```

最大值函数 (MAX)

返回列的最大值

```
SELECT MAX(列名) FROM 表名;
```

最小值函数 (MIN)

返回列的最小值

```
SELECT MIN(列名) FROM 表名;
```

求和函数 (SUM)

返回列的总和

```
SELECT SUM(列名) FROM 表名;
```

排除重复值

以上5个聚集函数都可以如下使用:

- 对所有的行执行计算, 指定ALL参数或不给参数(因为ALL是默认行为);
- 只包含不同的值, 指定DISTINCT参数。

ALL为默认 ALL参数不需要指定, 因为它是默认行为。如果 不指定DISTINCT, 则假定为ALL。

```
SELECT AVG(DISTINCT 列名) FROM 表名;
```

注意: 如果指定列名, 则DISTINCT只能用于COUNT()。DISTINCT 不能用于COUNT(*), 因此不允许使用COUNT(DISTINCT), 否则会产生错误。类似地, DISTINCT必须使用列名, 不能用于计算或表达式。

将**DISTINCT**用于**MIN()**和**MAX()** 虽然DISTINCT从技术上可用于MIN()和MAX(), 但这样做实际上没有价值。一个列中的最小值和最大值不管是否包含不同值都是相同的。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间: 2020-01-16 07:59:45

- 分组查询 (GROUP BY语句)
 - 分组查询
 - WITH ROLLUP
 - 过滤分组
 - 分组和排序(ORDERBY与GROUPBY)区别
 - SELECT子句顺序

分组查询 (GROUP BY语句)

分组查询

在具体使用GROUP BY子句前，需要知道一些重要的规定。

- 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，在建立分组时，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SELECT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，它们将分为一组。
- GROUP BY子句必须出现在WHERE子句之后，ORDER BY子句之前。

例：统计每个供应商的产品数量

```
SELECT vend_id,COUNT(prod_name) AS vent_prod_count FROM products GROUP BY vend.
```

Result Grid			
	vend_id	vent_count	
▶	1001	3	
	1002	2	
	1003	7	
	1005	2	

WITH ROLLUP

使用ROLLUP 使用WITH ROLLUP关键字，可以得到每个分组以及每个分组汇总级别(针对每个分组)的值

例：统计每个供应商的产品，并在最后一行返回一共有多少产品。

```
SELECT vend_id,COUNT(prod_name) AS vent_prod_count FROM products GROUP BY vend,
```

Result Grid			Filter
	vend_id	vent_count	
▶	1001	3	
	1002	2	
	1003	7	
	1005	2	
	NULL	14	

过滤分组

过滤分组不能用WHERE 子句，得用HAVING 操作符。**HAVING** 可以和 **WHERE** 连用

HAVING支持所有**WHERE**操作，本笔记有关WHERE的所有这些技术和选项都适用于 HAVING。它们的句法是相同的，只是关键字有差别。

HAVING和**WHERE**的差别，WHERE在数据分组前进行过滤，HAVING在数据分组后进行过滤。这是一个重要的区别，WHERE排除的行不包括在分组中。这可能会改变计算值，从而影响HAVING子句中基于这些值过滤掉的分组。

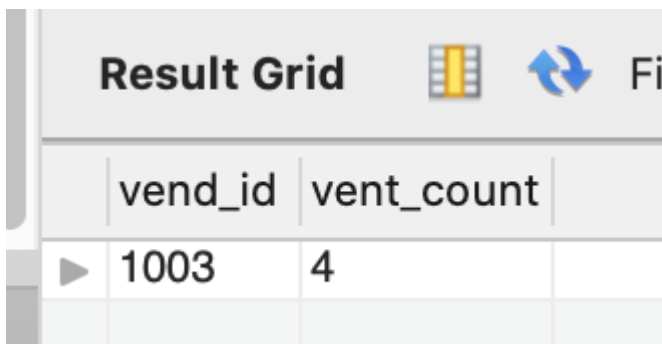
例：统计 产品数量大于2的供应商

```
SELECT vend_id,COUNT(prod_name) AS vent_prod_count FROM products GROUP BY vend,
```

Result Grid			Filter
	vend_id	vent_count	
▶	1001	3	
	1003	7	

例：统计 产品价格大于等于10，数量大于2的供应商

```
SELECT vend_id,COUNT(prod_name) AS vent_prod_count
FROM products
WHERE prod_price>=10
GROUP BY vend_id HAVING COUNT(prod_name)>2;
```



Result Grid	
vend_id	vent_count
1003	4

分组和排序(ORDERBY与GROUPBY)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顺序
任意列都可以使用(甚至非选择的列也可以使用)	只可能使用选择列或表达式列，而且必须使用每个选择列表表达式
不一定需要	如果与聚集函数一起使用列(或表达式)，则必须使用

SELECT子句顺序

关键词	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

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间：2020-01-16 17:56:46

- 子查询(IN)
 - 用子查询进行过滤
 - 作为计算字段使用子查询

子查询(IN)

子查询(subquery)，即嵌套在其他查询中的查询。

本文采用了书中的数据库来介绍。

表B-3 customers表的列

列	说 明
cust_id	唯一的顾客ID
cust_name	顾客名
cust_address	顾客的地址
cust_city	顾客的城市
cust_state	顾客的州
cust_zip	顾客的邮政编码
cust_country	顾客的国家
cust_contact	顾客的联系名
cust_email	顾客的联系email地址

表B-4 orders表的列

列	说 明
order_num	唯一订单号
order_date	订单日期
cust_id	订单顾客ID（关系到customers表的cust_id）

表B-5 orderitems表的列

列	说 明
order_num	订单号（关联到orders表的order_num）
order_item	订单物品号（在某个订单中的顺序）
prod_id	产品ID（关联到products表的prod_id）
quantity	物品数量
item_price	物品价格

订单存储在两个表中。对于包含订单号、客户ID、订单日期的每个订单，orders表存储一行。各订单的物品存储在相关的orderitems表中。orders表不存储客户信息。它只存储客户的ID。实际的客户信息存储在customers表中。

用子查询进行过滤

现在，假如需要列出订购物品TNT2的所有客户，应该怎样检索？下面列出具体的步骤。

1. 检索包含物品TNT2的所有订单的编号。

```
SELECT order_num FROM orderitems WHERE prod_id ='TNT2';
```

- 检索具有前一步骤列出的订单编号的所有客户的ID。

```
SELECT cust_id FROM orders WHERE order_num IN (20005,20007);
```

- 检索前一步骤返回的所有客户ID的客户信息

```
SELECT * FROM customers WHERE cust_id IN (10001,10004);
```

上述每个步骤都可以单独作为一个查询来执行。可以把一条SELECT 语句返回的结果用于另一条SELECT语句的WHERE子句。也可以使用子查询来把3个查询组合成一条语句。

子查询语法

```
SELECT * FROM 表名1 WHERE 列名1 IN (SELECT 列名2 FROM 表名2)
```

```
SELECT *
FROM customers
WHERE cust_id IN (
    SELECT cust_id FROM orders WHERE order_num IN
    (SELECT order_num FROM orderitems WHERE prod_id ='TNT2')
);
```

作为计算字段使用子查询

使用子查询的另一方法是创建计算字段。

假如需要显示customers表中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，遵循下面的步骤。

- 从customers表中检索客户列表。
- 对于检索出的每个客户，统计其在orders表中的订单数目。

```
SELECT cust_name,cust_state,
(SELECT COUNT(*) FROM orders WHERE orders.cust_id=customers.cust_id) AS orders
FROM customers;
```

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间： 2020-01-16 17:35:14

- 连接查询 (JOIN)
 - 内部联结 (INNER JOIN)
 - 联结多个表
 - 自联结
 - 自然联结
 - 外部联结
 - 使用带聚集函数的联结
 - 使用联结和联结条件

连接查询 (JOIN)

内部联结 (INNER JOIN)

又名等值联结(equijoin)

JOIN语句可以关联两张表，但是必须带上WHERE子句，否则返回结果是笛卡尔积。由于列会出现重名的情况，所以我们需要对重名的列，使用完全限定名。

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。检索出的行的数目将是第一个表中的行数乘以第二个表中的行数。

完全限定列名 在引用的列可能出现二义性时，必须使用完全限定列名(用一个点分隔的表名和列名)。如果引用一个 没有用表名限制的具有二义性的列名，MySQL将返回错误。

不要忘了WHERE子句 应该保证所有联结都有WHERE子句，否则MySQL将返回比想要的多得多数据。同理，应该保证WHERE子句的正确性。不正确的过滤条件将导致MySQL返回不正确的数据。



语法

注意：列1，2都是你自己指定的列，假定 表1.列1 = 表2.列1。

```
SELECT 表1.列1 , 表1.列2, 表2.列2
FROM 表1,表2
WHERE 表1.列1 = 表2.列1;
```

例：查询所有供应商及其产品详情

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

Result Grid   Filter Rows: <input type="text"/>			
	vend_name	prod_name	prod_price
▶	Anvils R Us	.5 ton anvil	5.99
	Anvils R Us	1 ton anvil	9.99
	Anvils R Us	2 ton anvil	14.99
	LT Supplies	Fuses	3.42
	LT Supplies	Oil can	8.99
	ACME	Detonator	13.00

内部联结，两个表之间的关系是由FROM子句的组成，以INNER JOIN指定。在使用这种语法时，联结条件用特定的ON子句而不是WHERE子句给出。传递给ON的实际条件与传递给WHERE的相同。



使用哪种语法 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，但是使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。

```
SELECT 表1.列1, 表1.列2, 表2.列2
FROM 表1 INNER JOIN 表2
ON 表1.列1 = 表2.列1;
```

例：查询所有供应商及其产品详情

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

结果和等值查询的一样，此语句中的SELECT与前面的SELECT语句相同，但FROM子句不同。这里，两个表之间的关系是FROM子句的组成部分，以INNER JOIN指定。在使用这种语法时，联结条件用特定的ON子句而不是WHERE子句给出。传递给ON的实际条件与传递给WHERE的相同。

Result Grid   Filter Rows: <input type="text"/>			
	vend_name	prod_name	prod_price
▶	Anvils R Us	.5 ton anvil	5.99
	Anvils R Us	1 ton anvil	9.99
	Anvils R Us	2 ton anvil	14.99
	LT Supplies	Fuses	3.42
	LT Supplies	Oil can	8.99
	ACME	Detonator	13.00

联结多个表

性能考虑 MySQL在运行时关联指定的每个表以处理联结。这种处理可能是非常耗费资源的，因此应该仔细，不要联结 不必要的表。联结的表越多，性能下降越厉害。

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。首先列出所有表，然后定义表之间的关系。

语法

注意：列1，2，3，4都是你自己指定的列，假定 表1.列1 = 表2.列1，表2.列4 = 表3.列4。

```
SELECT 表1.列1, 表1.列2, 表2.列2, 表3.列2
FROM 表1,表2,表3
WHERE 表1.列1 = 表2.列1 AND 表2.列4 =表3.列4
```

例：查询2005订单的产品数量详情

```
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;
```

	prod_name	vend_name	prod_price	quantity
▶	.5 ton anvil	Anvils R Us	5.99	10
	1 ton anvil	Anvils R Us	9.99	3
	TNT (5 sticks)	ACME	10.00	5
	Bird seed	ACME	10.00	1

自联结

自连接，指的是一个表自己和自己连接。

下面我们来看一个例子，假如你发现某物品(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也存在这些问题。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其他物品。

解决办法1（不推荐，子查询，效率低）

```
SELECT prod_id,prod_name
FROM products
WHERE vend_id =
(SELECT vend_id FROM products WHERE prod_id = 'DTNTR');
```

prod_id	prod_name
FC	Carrots
SAFE	Safe
SLING	Sling
TNT1	TNT (1 stick)
TNT2	TNT (5 sticks)

解决办法2（推荐，自连接）

```
SELECT p1.prod_id , p1.prod_name
FROM products AS p1, products AS p2
WHERE p1.vend_id = p2.vend_id AND p2.prod_id = 'DTNTR';
```

prod_id	prod_name
FC	Carrots
SAFE	Safe
SLING	Sling
TNT1	TNT (1 stick)
TNT2	TNT (5 sticks)

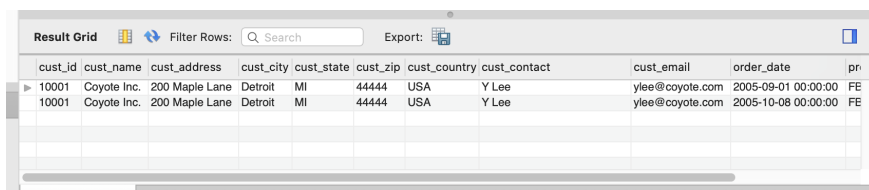
WHERE(通过匹配p1中的 vend_id和p2中的vend_id)首先联结两个表，然后按第二个表中的 prod_id过滤数据，返回所需的数据。

自然联结

无论何时对表进行联结，应该至少有一个列出现在不止一个表中(被联结的列)。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。自然联结是这样一种联结，其中你只能选择那些唯一的列。这一般是通过表使用通配符(SELECT *)，对所有其他表的列使用明确的子集来完成的。

下面举一个例子：

```
SELECT c.* , o.order_date,oi.prod_id,oi.quantity,oi.item_price
FROM customers AS c , orders AS o , orderitems AS oi
WHERE c.cust_id =o.cust_id
AND oi.order_num=o.order_num
AND prod_id='FB';
```



cust_id	cust_name	cust_address	cust_city	cust_state	cust_zip	cust_country	cust_contact	cust_email	order_date	pr
10001	Coyote Inc.	200 Maple Lane	Detroit	MI	44444	USA	Y Lee	ylee@coyote.com	2005-09-01 00:00:00	FE
10001	Coyote Inc.	200 Maple Lane	Detroit	MI	44444	USA	Y Lee	ylee@coyote.com	2005-10-08 00:00:00	FE

在这个例子中，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。



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不同。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。

例如，可能需要使用联结来完成以下工作：

- 对每个客户下了多少订单进行计数，包括那些至今尚未下订单的客户；
- 列出所有产品以及订购数量，包括没有人订购的产品；
- 计算平均销售规模，包括那些至今尚未下订单的客户。

例如，检索所有客户，包括那些没有订单的客户

```
SELECT c.cust_id,o.order_num
FROM customers AS c LEFT OUTER JOIN orders AS o
ON c.cust_id =o.cust_id;
```

Result Grid  		
	cust_id	order_num
▶	10001	20005
	10001	20009
	10002	NULL
	10003	20006
	10004	20007
	10005	20008
	10007	NULL
	10008	NULL

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)。

但是, 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LEFT 指出的是OUTER JOIN左边的表)。

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。为了从右边的表中选择所有行, 应该使用RIGHT OUTER JOIN。

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不同。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。

使用带聚集函数的联结

例如: 如果要检索所有客户及每个客户所下的订单数

注意: 这里某些客户可能没有下单, 我们的需求是查询所有客户, 包括哪些没有下单的客户, 所以使用左联

```
SELECT customers.cust_name,customers.cust_id,
COUNT(orders.order_num) AS cust_ordersum
FROM customers LEFT OUTER JOIN orders
ON customers.cust_id=orders.cust_id
GROUP BY customers.cust_id;
```

	cust_name	cust_id	cust_ordersum	
▶	Coyote Inc.	10001	2	
	Mouse House	10002	0	
	Wascals	10003	1	
	Yosemite Place	10004	1	
	E Fudd	10005	1	
	杰威尔	10007	0	

使用联结和联结条件

- 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。
- 保证使用正确的联结条件，否则将返回不正确的数据。
- 应该总是提供联结条件，否则会得出笛卡儿积。
- 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。虽然这样做是合法的，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。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间：2020-01-16 15:36:25

- [组合查询\(UNION\)](#)

组合查询(UNION)

MySQL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)或复合查询(compound query)。

有两种基本情况, 其中需要使用组合查询:

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

组合查询和多个WHERE条件 多数情况下, 组合相同表的两个查询完成的工作与具有多个WHERE子句条件的单条查询完成的 工作相同。换句话说, 任何具有多个WHERE子句的SELECT语句 都可以作为一个组合查询给出, 在以下段落中可以看到这一点。这两种技术在不同的查询中性能也不同。因此, 应该试一下这 两种技术, 以确定对特定的查询哪一种性能更好。

语法

```
SELECT * FROM 表1 WHERE 列1 < 3
UNION
SELECT * FROM 表1 WHERE 列1 > 5;
```

等价语法

```
SELECT * FROM 表1 WHERE 列1 < 3 OR 列1 > 5;
```

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间: 2020-01-16 17:30:05

- 插入数据(INSERT)
 - 插入完整的行
 - 插入多个行
 - 插入检索出的数据

插入数据(INSERT)

没有输出 INSERT语句一般不会产生输出。

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:

- 插入完整的行;
- 插入行的一部分;
- 插入多行;
- 插入某些查询的结果。

插入完整的行

```
# 不安全的做法
INSERT INTO 表名 VALUES ('列值1','列值2','列值3');

# 安全的做法
INSERT INTO 表名(列名1, 列名2, 列名3) VALUES ('列值1','列值2','列值3');
```

虽然这种语法很简单, 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, 并且还依赖于其次序容易获得的信息。即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。因此, 编写依赖于特定列次序的SQL语句是很不安全的。如果这样做, 有时难免会出问题。

总是使用列的列表 一般不要使用没有明确给出列的列表的 INSERT语句。使用列的列表能使SQL代码继续发挥作用, 即使 表结构发生了变化。

仔细地给出值 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, 则必须给每个表列提供一个值。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, 相应的行插入不成功。

省略列 如果表的定义允许, 则可以在INSERT操作中省略某些列。省略的列必须满足以下某个条件。

- 该列定义为允许NULL值(无值或空值)。
- 在表定义中给出默认值。这表示如果不给出值, 将使用默认值。

如果对表中不允许NULL值且没有默认值的列不给出值, 则 MySQL将产生一条错误消息, 并且相应的行插入不成功。

插入多个行

只要每条INSERT语句中的列名(和次序)相同, 可以如下组合各语句:

```
INSERT INTO 表名(列名1, 列名2, 列名3) VALUES ('列值1','列值2','列值3'), ('列值4','列值5','列值6');
```

其中单条INSERT语句有多组值，每组值用一对圆括号括起来，用逗号分隔。

提高INSERT的性能 此技术可以提高数据库处理性能，因为MySQL用单条INSERT语句处理多个插入比使用多条INSERT语句快。

插入检索出的数据

INSERT一般用来给表插入一个指定列值的行。

但是，INSERT还存在另一种形式，可以利用它将一条SELECT语句的结果插入表中。

这就是所谓的INSERT SELECT，顾名思义，它是由一条INSERT语句和一条SELECT语句组成的。

```
INSERT INTO customers[表名](列名1, 列名2, 列名3) SELECT (列名1, 列名2, 列名3) FROM custnew;
```

这个例子使用INSERT SELECT从custnew中将所有数据导入customers。

SELECT语句从custnew检索出要插入的值，而不是列出它们。

SELECT中列出的每个列对应于customers表名后所跟的列表中的每个列。

这条语句将插入多少行有赖于custnew表中有多少行。如果这个表为空，则没有行被插入(也不产生错误，因为操作仍然是合法的)。

如果这个表确实含有数据，则所有数据将被插入到customers。

这个例子导入了cust_id(假设你能够确保cust_id的值不重复)。你也可以简单地省略这列(从INSERT和SELECT中)，这样MySQL就会生成新值。

INSERTSELECT中的列名 为简单起见，这个例子在INSERT和SELECT语句中使用了相同的列名。但是，不一定要列名匹配。事实上，MySQL甚至不关心SELECT返回的列名。它使用的是列的位置，因此SELECT中的第一列(不管其列名)将用来填充表列中指定的第一个列，第二列将用来填充表列中指定的第二个列，如此等等。这对于从使用不同列名的表中导入数据是非常有用的。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间：2020-03-27 00:43:01

- 更新数据(UPDATE)
 - 更新单个列数据
 - 更新多个列数据
 - 删除某个列的值
 - 更新所有数据
 - 指导原则

更新数据(UPDATE)

没有输出 UPDATE语句一般不会产生输出。

为了从一个表中删除(去掉)数据，使用UPDATE语句。可以两种方式使用UPDATE:

- 更新表中特定行;
- 更新表中所有行。

不要省略WHERE子句 在使用UPDATE时一定要细心。因为稍不注意，就会错误地更改表中所有行。

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现一个错误，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)。

更新单个列数据

```
UPDATE 表名 SET 列名1 = '新值' WHERE 列名1='值';
```

更新多个列数据

```
UPDATE 表名 SET 列名1 = '新值',列名2 = '新值' WHERE 列名1='值';
```

删除某个列的值

```
UPDATE 表名 SET 列名1 = NULL WHERE 列名1='值';
```

NULL无需加引号!

更新所有数据

如果要更新表中的所有数据，可以不加WHERE条件! 注意，数据无价，谨慎操作，不能撤销!

语法 (不推荐，在MySQLWorkbench 可能会报错，因为MySQLWorkbench默认配置UPDATE语句必须WHERE 子句)

```
UPDATE 表名 SET 列名1 = '新值';
```

指导原则

如果省略了WHERE子句，则UPDATE或DELETE将被应用到表中 190 所有的行。换句话说，如果执行UPDATE而不带WHERE子句，则表中每个行都将用新值更新。类似地，如果执行DELETE语句而不带WHERE子句，表的所有数据都将被删除。

下面是许多SQL程序员使用UPDATE或DELETE时所遵循的习惯。

- 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语句。
- 保证每个表都有主键(如果忘记这个内容，请参阅第15章)，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)。
- 在对UPDATE或DELETE语句使用WHERE子句前，应该先用SELECT进行测试，保证它过滤的是正确的记录，以防编写的WHERE子句不正确。
- 使用强制实施引用完整性的数据库(关于这个内容，请参阅第15 章)，这样MySQL将不允许删除具有与其他表相关联的数据的行。

小心使用 MySQL没有撤销(undo)按钮。应该非常小心地 使用UPDATE和DELETE，否则你会发现自己更新或删除了错误 的数据。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间： 2020-01-16 19:43:11

- 删除数据(DELETE)
 - 删除指定行数据
 - 删除所有数据
 - 指导原则

删除数据(DELETE)

没有输出 DELETE语句一般不会产生输出。

为了从一个表中删除(去掉)数据，使用DELETE语句。可以两种方式使用DELETE:

- 从表中删除特定的行;
- 从表中删除所有行。

不要省略WHERE子句 在使用DELETE时一定要细心。因为稍不注意，就会错误地删除表中所有行。

删除表的内容而不是表 DELETE语句从表中删除行，甚至是 删除表中所有行。但是，DELETE不删除表本身。

删除指定行数据

```
DELETE FROM 表名 WHERE 列名1='值'
```

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UPDATE语句。

删除所有数据

如果想从表中删除所有行，不要使用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)。

语法1（推荐，效率高）

```
TRUNCATE TABLE 表名;
```

语法2（不推荐，效率低，在MySQLWorkbench 可能会报错，因为MySQLWorkbench默认配置DELETE语句必须WHERE 子句）

```
DELETE FROM 表名;
```

指导原则

如果省略了WHERE子句，则UPDATE或DELETE将被应用到表中 190 所有的行。换句话说，如果执行UPDATE而不带WHERE子句，则表中每个行都将用新值更新。类似地，如果执行DELETE语句而不带WHERE子句，表的所有数据都将被删除。

下面是许多SQL程序员使用UPDATE或DELETE时所遵循的习惯。

- 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语句。
- 保证每个表都有主键(如果忘记这个内容，请参阅第15章)，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)。
- 在对UPDATE或DELETE语句使用WHERE子句前，应该先用SELECT进行测试，保证它过滤的是正确的记录，以防编写的WHERE子句不正确。
- 使用强制实施引用完整性的数据库(关于这个内容，请参阅第15章)，这样MySQL将不允许删除具有与其他表相关联的数据的行。

小心使用 MySQL没有撤销(undo)按钮。应该非常小心地使用UPDATE和DELETE，否则你会发现你更新或删除了错误的数据库。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间： 2020-01-16 19:27:27

- 增删改表(CREATE、ALTER、DROP)
 - 创建表
 - 指定非空列
 - 主键
 - 使用AUTO_INCREMENT (自增)
 - 指定默认值
 - 更新表
 - 删除表
 - 重命名表

增删改表(CREATE、ALTER、DROP)

没有输出 CREATE、ALTER、DROP语句一般不会产生输出。

创建表

语法

```
CREATE TABLE 表名(
    列名1 INT PRIMARY KEY,
    列名2 INT
);
```

处理现有的表 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要求首先手工删除该表，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的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。

指定非空列

NOT NULL,语法

```
CREATE TABLE 表名(
    列名1 INT PRIMARY KEY,
    列名2 INT NOT NULL
);
```

NULL值就是没有值或缺值。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值的列不接受该列没有值的行，换句话说，在插入或更新行时，该列必须有值，否则插入失败。

理解NULL 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，其间没有字符)，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

主键和NULL值 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值的列不能作为唯一标识。

语法1

```
CREATE TABLE 表名(
    列名1 INT PRIMARY KEY,
    列名2 INT NOT NULL
);
```

语法2

```
CREATE TABLE 表名(
    列名1 INT ,
    列名2 INT NOT NULL ,
    PRIMARY KEY(列名1)
);
```

多个列作主键 语法

```
CREATE TABLE 表名(
    列名1 INT ,
    列名2 INT NOT NULL ,
    PRIMARY KEY(列名1,列名2)
);
```

使用AUTO_INCREMENT（自增）

语法

```
CREATE TABLE 表名(
    列名1 INT PRIMARY KEY AUTO_INCREMENT,
    列名2 INT NOT NULL
);
```

确定AUTO_INCREMENT值 让MySQL生成(通过自动增量)主 键的一个缺点是你不知道这些值都是谁。

考虑这个场景:你正在增加一个新订单。这要求在orders表 中创建一行，然后在orderitems表中对订购的每项物品创建一 行。order_num在orderitems表中与订单细节一起存储。这 就是为什么orders表和orderitems表为相互关联的表的原 因。这显然要求你在插入orders行之后，插入orderitems行 之前知道生成的order_num。

那么，如何在使用AUTO_INCREMENT列时获得这个值呢?可使用last_insert_id()函数获得这个值，如下所示:

```
SELECT last_insert_id();
```

此语句返回最后一个AUTO_INCREMENT值，然后可以将它用于后续的MySQL语句。

指定默认值

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语句的列定义中的DEFAULT关键字指定。

语法

```
CREATE TABLE 表名(
    列名1 INT PRIMARY KEY AUTO_INCREMENT,
    列名2 INT DEFAULT 0
);
```

不允许函数 与大多数DBMS不一样，MySQL不允许使用函数作为默认值，它只支持常量。

使用默认值而不是NULL值 许多数据库开发人员使用默认值而不是NULL列，特别是对用于计算或数据分组的列更是如此。

更新表

小心使用ALTER TABLE 使用ALTER TABLE要极为小心，应该在进行改动前做一个完整的备份(模式和数据的备份)。数据库表的更改不能撤销，如果增加了不需要的列，可能不能删除它们。类似地，如果删除了不应该删除的列，可能会丢失该列中的所有数据。

添加列

```
ALTER TABLE 表名 ADD 列名 INT;
```

删除列

```
ALTER TABLE 表名 DROP COLUMN 列名 INT;
```

ALTER TABLE的一种常见用途是定义外键

```
ALTER TABLE 表名
ADD CONSTRAINT 外键名字
FOREIGN KEY (列名) REFERENCES 外键表(外键表的主键的列名);
```

删除表

删除表(删除整个表而不是其内容)非常简单，使用DROP TABLE语句即可

谨慎操作，删除表后，表中的数据全部删除完成，无法撤销。

```
DROP TABLE 表名;
```

重命名表

使用RENAME TABLE语句可以重命名一个表:

```
RENAME TABLE 旧表名 TO 新表名 ;
```

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间: 2020-01-16 21:25:08

- [MySQL 数据类型](#)
 - [串数据类型](#)
 - [数值数据类型](#)
 - [日期和时间数据类型](#)
 - [二进制数据类型](#)

MySQL 数据类型

数据类型是定义列中可以存储什么数据以及该数据实际怎样存储的基本规则。

数据类型用于以下目的。

- 数据类型允许限制可存储在列中的数据。例如，数值数据类型列 只能接受数值。
- 数据类型允许在内部更有效地存储数据。可以用一种比文本串更 简洁的格式存储数值和日期时间值。
- 数据类型允许变换排序顺序。如果所有数据都作为串处理，则1 位于10之前，而10又位于2之前(串以字典顺序排序，从左边开始 比较，一次一个字符)。作为数值数据类型，数值才能正确排序。

串数据类型

数据类型	说明
CHAR	1~255个字符的定长串。它的长度必须在创建时指定，否则MySQL 假定为CHAR(1)
ENUM	接受最多64 K个串组成的一个预定义集合的某个串
LONGTEXT	与TEXT相同，但最大长度为4 GB
MEDIUMTEXT	与TEXT相同，但最大长度为16 K
SET	接受最多64个串组成的一个预定义集合的零个或多个串
TEXT	最大长度为64 K的变长文本
TINYTEXT	与TEXT相同，但最大长度为255字节
VARCHAR	长度可变，最多不超过255字节。如果在创建时指定为 VARCHAR(n)，则可存储0到n个字符的变长串(其中 n≤255)

使用引号 不管使用何种形式的串数据类型，串值都必须括在引号内(通常单引号更好)。

当数值不是数值时 你可能会认为电话号码和邮政编码应该 存储在数值字段中(数值字段只存储数值数据)，但是，这样做却是不可取的。如果在数值字段中存储邮政编码01234，则 保存的将是数值1234，实际上丢失了一位数字。需要遵守的基本规则是:如果数值是计算(求和、平均等)中使用的数值，则应该存储在数值数据类型列中。如果作为字符串(可能只包含数字)使用，则应该保存在串数据类型列中。

数值数据类型

说明	说明
BIT	位字段，1~64位。(在MySQL 5之前，BIT在功能上等价于 TINYINT)
BIGINT	整数值，支持-9223372036854775808~9223372036854775807 (如果是UNSIGNED，为0~18446744073709551615)的数
BOOLEAN(或 BOOL)	布尔标志，或者为0或者为1，主要用于开/关(on/off)标志
DECIMAL(或 DEC)	精度可变的浮点值
DOUBLE	双精度浮点值
FLOAT	单精度浮点值
INT(或 INTEGER)	整数值，支持-2147483648~2147483647 (如果是UNSIGNED，为0~4294967295)的数
MEDIUMINT	整数值，支持-8388608~8388607 (如果是UNSIGNED，为0~16777215)的数
REAL	4字节的浮点值
SMALLINT	整数值，支持-32768~32767 (如果是UNSIGNED，为0~65535)的数
TINYINT	整数值，支持-128~127 (如果为UNSIGNED，为0~255)的数

不使用引号 与串不一样，数值不应该括在引号内。

存储货币数据类型 MySQL中没有专门存储货币的数据类型，一般情况下使用 DECIMAL(8, 2)

BOOLEAN的陷阱 这个类型会被MySQL自动转换为TINYINT(1)，而且可以插入除了0和1以外的值，请直接用TINYINT(1)。

日期和时间数据类型

MySQL使用专门的数据类型来存储日期和时间值

数据类型	说明
DATE	表示1000-01-01~9999-12-31的日期，格式为 YYYY-MM-DD
DATETIME	DATE和TIME的组合
TIMESTAMP	功能和DATETIME相同(但范围较小)
TIME	格式为HH:MM:SS
YEAR	用2位数字表示，范围是70(1970年)~69(2069 年)，用4位数字表示，范围是1901年~2155年

二进制数据类型

二进制数据类型可存储任何数据(甚至包括二进制信息)，如图像、多媒体、字处理文档等

数据类型	说明
BLOB	Blob最大长度为64 KB
MEDIUMBLOB	Blob最大长度为16 MB
LONGBLOB	Blob最大长度为4 GB
TINYBLOB	Blob最大长度为255字节

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间： 2020-01-16 20:19:47

- 视图(VIEW)

视图(VIEW)

视图的一些常见应用。

- 重用SQL语句。
- 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。
- 使用表的组成部分而不是整个表。
- 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。
- 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。

性能问题 因为视图不包含数据，所以每次使用视图时，都必须处理查询执行时所需的任一个检索。如果你用多个联结和过滤创建了复杂的视图或者嵌套了视图，可能会发现性能下降得很厉害。因此，在部署使用了大量视图的应用前，应该进行测试。

将视图用于检索 一般，应该将视图用于检索(SELECT语句) 而不适用于更新(INSERT、UPDATE和DELETE)。

视图的最常见的应用之一是隐藏复杂的SQL，这通常都会涉及联结。请看下面的例子：

查询所有的购买过商品是顾客，分别需要查询3张表

```
SELECT cust_name,cust_contact,prod_id
FROM customers,orders,orderitems
WHERE customers.cust_id=orders.cust_id
AND orderitems.order_num=orders.order_num;
```

我们需要写那么长的SQL语句，用了视图

```
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=orders.order_num;
```

以后我们的查询语句，就只需要一句话

```
SELECT * FROM productcustomers;
```

这个视图我们可以看做是特殊的表，也能用WHERE过滤

```
SELECT * FROM productcustomers WHERE prod_id = 'TNT2';
```

WHERE子句与WHERE子句 如果从视图检索数据时使用了一条 WHERE子句，则两组子句(一组在视图中，另一组是传递给视图的)将自动组合。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间: 2020-01-20 17:14:29

- 存储过程(PROCEDURE)
 - 存储过程的优缺点
 - 优点
 - 缺点
 - 使用存储过程
 - 执行存储过程
 - 创建存储过程
 - 删除存储过程
 - 使用参数
 - 检查存储过程

存储过程(PROCEDURE)

存储过程，就是提前一组提前编写好的SQL语句，能够提高性能。

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。

存储过程的优缺点

优点

- 通过把处理封装在容易使用的单元中，简化复杂的操作(正如前面例子所述)。
- 由于不要求反复建立一系列处理步骤，这保证了数据的完整性。如果所有开发人员 and 应用程序都使用同一(试验和测试)存储过程，则所使用的代码都是相同的。这一点的延伸就是防止错误。需要执行的步骤越多，出错的可能性就越大。防止错误保证了数据的一致性。
- 简化对变动的管理。如果表名、列名或业务逻辑(或别的内容)有变化，只需要更改存储过程的代码。使用它的人员甚至不需要知道这些变化。
- 提高性能。因为使用存储过程比使用单独的SQL语句要快。
- 存在一些只能用在单个请求中的MySQL元素和特性，存储过程可以使用它们来编写功能更强更灵活的代码

换句话说，使用存储过程有3个主要的好处，即**简单、安全、高性能**。

缺点

- 一般来说，存储过程的编写比基本SQL语句复杂，编写存储过程需要更高的技能，更丰富的经验。
- 你可能没有创建存储过程的安全访问权限。许多数据库管理员限制存储过程的创建权限，允许用户使用存储过程，但不允许他们创建存储过程。

在实际开发中，一般不用存储过程（来自阿里的Java开发规范）。

使用存储过程

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接受存储过程的名字以及需要传递给它的任意参数。

```
CALL productpricing (@pricelow,@pricehigh,@priceaverage)
```

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。

创建存储过程

请看一个例子——一个返回产品平均价格的存储过程。

以下是其代码：

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

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。此存储过程没有参数，但后跟的()仍然需要。

BEGIN和END语句用来限定存储过程体，过程体本身仅是一个简单的SELECT语句。

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，因为这段代码并未调用存储过程，这里只是为以后使用而创建它。

如何使用这个存储过程

```
CALL productpricing();
```

CALL productpricing();执行刚创建的存储过程并显示返回的结果。

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)。

MySQL客户端的分隔符

默认的MySQL语句分隔符为；(正如你已经在迄今为止所使用的MySQL语句中所看到的那样)。mysql客户端也使用；作为语句分隔符。如果mysql客户端要解释存储过程自身内的；字符，则它们最终不会成为存储过程的成分，这会使存储过程中的SQL出现句法错误。

解决办法是临时更改命令行实用程序的语句分隔符

```

DELIMITER //
CREATE PROCEDURE productpricing()
BEGIN
    SELECT * Avg(prod_price) AS priceaverage FROM products;
END //
DELIMITER ;

```

DELIMITER // 告诉客户端使用 // 作为新的语句结束分隔符，可以看到标志存储过程结束的END定义为 END // 而不是 END；。

这样，存储过程体内的；仍然保持不动，并且正确地传递给数据库引擎。最后，为恢复为原来的语句分隔符，可使用 DELIMITER ；。

除 \ 符号外，任何字符都可以用作语句分隔符。

删除存储过程

存储过程在创建之后，被保存在服务器上以供使用，直至被删除。DROP命令从服务器中删除存储过程。

```
DROP PROCEDURE productpricing;
```

这条语句删除刚创建的存储过程。请注意没有使用后面的()，只给出存储过程名。

仅当存在时删除

如果指定的过程不存在，则DROP PROCEDURE 将产生一个错误。当过程存在想删除它时(如果过程不存在也不产生错误)可使用**DROP PROCEDURE IF EXISTS**。

```
DROP PROCEDURE IF EXISTS productpricing;
```

使用参数

参数分为 输入参数 (IN 关键字)，输出参数 (OUT 关键字)

参数的数据类型

存储过程的参数允许的数据类型与表中使用的数据类型相同。注意，记录集不是允许的类型，因此，不能通过一个参数返回多个行和列。

变量名 所有MySQL变量都必须以@开始。

下面是一个例子，ordertotal接受订单号并返回该订单的合计

```

CREATE PROCEDURE ordertotal(
    IN onumber INT,
    OUT ototal DECIMAL(8,2)
)
BEGIN
    SELECT Sum(item_price * quantity)
    FROM orderitems
    WHERE order_num = onumber
    INTO ototal;
END;

```

此存储过程接受1个参数:onenumber 订单号, 每个参数必须具有指定的类型, 这里使用十进制值

onenumber定义为IN, 因为订单号被传入存储过程。ototal定义为OUT, 因为要从存储过程返回合计。

SELECT语句使用这两个参数, WHERE子句使用onenumber选择正确的行, INTO使用ototal存储计算出来的合计。

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)。

MySQL支持IN(传递给存储过程)、OUT(从存储过程传出, 如这里所用)和INOUT(对存储过程传入和传出)类型的参数。

存储过程的代码位于BEGIN和END语句内, 如前所见, 它们是一系列 SELECT语句, 用来检索值, 然后保存到相应的变量(通过指定INTO关键字)。

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。

可如下进行, 必须给ordertotal传递两个参数;第一个参数为订单号, 第二个参数为包含计算出来的合计的变量名。

```
CALL ordertotal (20005,@total);
# 显示合计
SELECT @total;
```

检查存储过程

为显示用来创建一个存储过程的CREATE语句, 使用SHOW CREATE PROCEDURE语句:

```
SHOW CREATE PROCEDURE ordertotal;
# 输出存储过程的创建语句
SHOW PROCEDURE STATUS;
# 显示所有的存储过程
```

SHOW PROCEDURE STATUS; 可以列出所有存储过程。为限制其输出, 可使用LIKE指定一个过滤模式, 例如:

```
SHOW PROCEDURE STATUS LIKE 'ordertotal';
```

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间: 2020-03-26 11:40:04

- 触发器(TRIGGER)
 - 创建触发器
 - 删除触发器
 - 使用触发器
 - INSERT触发器
 - DELETE触发器
 - UPDATE触发器

触发器(TRIGGER)

触发器是MySQL响应以下任意语句而自动执行的一条MySQL语句(或位于BEGIN和END语句之间的一组语句);

- DELETE
- INSERT
- UPDATE

其他MySQL语句不支持触发器。

创建触发器

在创建触发器时, 需要给出4条信息:

- 唯一的触发器名;
- 触发器关联的表;
- 触发器应该响应的活动(DELETE、INSERT或UPDATE);
- 触发器何时执行(处理之前或之后)。

保持每个数据库的触发器名唯一 最好是在数据库范围内使用唯一的触发器名。

触发器用CREATE TRIGGER语句创建。下面是一个简单的例子:

```
CREATE TRIGGER newproduct AFTER INSERT on products
FOR EACH ROW SELECT 'Product added';
```

CREATE TRIGGER用来创建名为newproduct的新触发器。

触发器可在一个操作发生之前或之后执行, 这里给出了AFTER INSERT, 所以此触发器将在INSERT语句成功执行后执行。

这个触发器还指定FOR EACH ROW, 因此代码对每个插入行执行。

在这个例子中, 文本Product added将对每个插入的行显示一次。为了测试这个触发器, 使用INSERT语句添加一行或多行到products 中, 你将看到对每个成功的插入, 显示Product added消息。

仅支持表 只有表才支持触发器, 视图不支持(临时表也不支持)。

触发器按每个表每个事件每次地定义, 每个表每个事件每次只允许一个触发器。因此, 每个表最多支持6个触发器(每条INSERT、UPDATE和DELETE的之前和之后)。单一触发器不能与多个事件或多个表关联, 所以, 如果你需要一个对INSERT和UPDATE操作执行的触发器, 则应该定义两个触发器。

触发器失败 如果BEFORE触发器失败，则MySQL将不执行请求的操作。此外，如果BEFORE触发器或语句本身失败，MySQL 将不执行AFTER触发器(如果有的话)。

删除触发器

删除一个触发器，可使用DROP TRIGGER语句

```
DROP TRIGGER newproduct;
```

触发器不能更新或覆盖。为了修改一个触发器，必须先删除它，然后再重新创建。

使用触发器

INSERT触发器

INSERT触发器在INSERT语句执行之前或之后执行。需要知道以下几点:

- 在INSERT触发器代码内，可引用一个名为NEW的虚拟表，访问被 插入的行;
- 在BEFORE INSERT触发器中，NEW中的值也可以被更新(允许更改 被插入的值);
- 对于AUTO_INCREMENT列，NEW在INSERT执行之前包含0，在INSERT 执行之后包含新的自动生成值。

下面举一个例子(一个实际有用的例子)。AUTO_INCREMENT列具有 MySQL自动赋予的值。

```
CREATE TRIGGER neworder AFTER INSERT ON orders
FOR EACH ROW SELECT NEW.order_num;
```

此代码创建一个名为neworder的触发器，它按照AFTER INSERT ON orders执行。在插入一个新订单到orders表时，MySQL生成一个新订单号并保存到order_num中。触发器从NEW. order_num取得这个值并返回它。

此触发器必须按照AFTER INSERT执行，因为在BEFORE INSERT语句执行之前，新order_num还没有生成。对于orders的每次插入使用这个触发器将总是返回新的订单号。

为测试这个触发器，试着插入一下新行，如下所示:

输入

```
INSERT INTO orders(order_date, cust_id)
VALUES(Now(), 10001);
```

输出

```
+-----+
| order_num |
+-----+
|      20010 |
+-----+
```

分析

orders 包含 3 个列。order_date 和 cust_id 必须给出，order_num由MySQL自动生成，而现在order_num还自动被返回。

BEFORE或AFTER? 通常，将BEFORE用于数据验证和净化(目的是保证插入表中的数据确实是需要的数据)。本提示也适用于UPDATE触发器。

DELETE触发器

DELETE触发器在DELETE语句执行之前或之后执行。需要知道以下两点:

- 在DELETE触发器代码内，你可以引用一个名为OLD的虚拟表，访问被删除的行;
- OLD中的值全都是只读的，不能更新。

下面的例子演示使用OLD保存将要被删除的行到一个存档表中:

```
CREATE TRIGGER deleteorder BEFORE DELETE ON orders
FOR EACH ROW
BEGIN
    INSERT INTO archive_orders(order_num,order_date,cust_id)
    VALUES (OLD.order_num,OLD.order_date,OLD.cust_id);
END;
```

在任意订单被删除前将执行此触发器。它使用一条INSERT语句将OLD中的值(要被删除的订单)保存到一个名为archive_orders的存档表中(为实际使用这个例子，你需要用与orders相同的列 创建一个名为archive_orders的表)。

使用BEFORE DELETE触发器的优点(相对于AFTER DELETE触发器 来说)为，如果由于某种原因，订单不能存档，DELETE本身将被放弃。

多语句触发器 触发器deleteorder使用BEGIN和 END语句标记触发器体。这在此例子中并不是必需的，不过也没有害处。使用BEGIN END块的好处是触发器能容纳多条SQL语句(在BEGIN END块中一条挨着一条)。

UPDATE触发器

UPDATE触发器在UPDATE语句执行之前或之后执行。需要知道以下几点:

- 在UPDATE触发器代码中，你可以引用一个名为OLD的虚拟表访问 以前 (UPDATE语句前)的值，引用一个名为NEW的虚拟表访问新更新的值;
- 在BEFORE UPDATE触发器中，NEW中的值可能也被更新(允许更改将要用于UPDATE语句中的值);
- OLD中的值全都是只读的，不能更新。

下面的例子保证州名缩写总是大写(不管UPDATE语句中给出的是大写还是小写):

```
CREATE TRIGGER updatevendor BEFORE UPDATE ON vendors
FOR EACH ROW SET NEW.vend_state= Upper(NEW.vend_state);
```

任何数据净化都需要在UPDATE语句之前进行，像这个例子中一样。每次更新一个行时，NEW.vend_state中的值(将用来更新表行的值)都用Upper(NEW.vend_state)替换。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间: 2020-03-27 00:46:09

- 事务(TRANSACTION)
 - 开始事务
 - 使用ROLLBACK
 - 使用COMMIT
 - 使用保留点
 - 更改默认的提交行为

事务(TRANSACTION)

事务处理(transaction processing)可以用来维护数据库的完整性, 它 保证成批的MySQL操作要么完全执行, 要么完全不执行。

MySQL仅InnoDB引擎支持事务!

事务处理是一种机制, 用来管理必须成批执行的MySQL操作, 以保证数据库不包含不完整的操作结果。利用事务处理, 可以保证一组操作不会中途停止, 它们 或者作为整体执行, 或者完全不执行(除非明确指示)。如果没有错误发 生, 整组语句提交给(写到)数据库表。如果发生错误, 则进行回退(撤销)以恢复数据库到某个已知且安全的状态。

在使用事务和事务处理时, 有几个关键词汇反复出现。下面是关于 事务处理需要知道的几个术语

- **事务(transaction)** 指一组SQL语句;
- **回退(rollback)** 指撤销指定SQL语句的过程;
- **提交(commit)** 指将未存储的SQL语句结果写入数据库表;
- **保留点(savepoint)** 指事务处理中设置的临时占位符(place-holder), 你可以对它发布回退(与回退整个事务处理不同)。

开始事务

```
START TRANSACTION
```

使用ROLLBACK

MySQL的ROLLBACK命令用来回退(撤销)MySQL语句, 请看下面的 语句:

```
START TRANSACTION;  
DELETE FROM ordertotals;  
SELECT * FROM ordertotals;  
ROLLBACK;  
SELECT * FROM ordertotals;
```

首先执行一条SELECT以显示该表不为空。然后开始一个事务处理, 用一条DELETE语句删除ordertotals中的所有行。另一条 SELECT语句验证ordertotals确实为空。这时用一条ROLLBACK语句回退 START TRANSACTION之后的所有语句, 最后一条SELECT语句显示该表不为空。

显然，**ROLLBACK**只能在一个事务处理内使用(在执行一条**START TRANSACTION**命令之后)。

哪些语句可以回退？

事务处理用来管理**INSERT**、**UPDATE**和 **DELETE**语句。

你不能回退**SELECT**语句。(这样做也没有什么意义。)

你不能回退**CREATE**或**DROP**操作。事务处理块中可以使用这两条语句，但如果你执行回退，它们不会被撤销

使用COMMIT

一般的MySQL语句都是直接针对数据库表执行和编写的。这就是 所谓的隐含提交(implicit commit)，即提交(写或保存)操作是自动进行的。

但是，在事务处理块中，提交不会隐含地进行。为进行明确的提交，使用**COMMIT**语句，如下所示：

```
START TRANSACTION;
DELETE FROM orderitems WHERE order_num=20010;
DELETE FROM orders WHERE order_num=20010;
COMMIT;
```

在这个例子中，从系统中完全删除订单20010。因为涉及更新

两个数据库表**orders**和**orderItems**，所以使用事务处理块来 保证订单不被部分删除。最后的**COMMIT**语句仅在不出错时写出更改。如果第一条**DELETE**起作用，但第二条失败，则**DELETE**不会提交(实际上，它是被自动撤销的)。

隐含事务关闭 当**COMMIT**或**ROLLBACK**语句执行后，事务会自动关闭(将来的更改会隐含提交)。

使用保留点

简单的**ROLLBACK**和**COMMIT**语句就可以写入或撤销整个事务处理。

但是，只是对简单的事务处理才能这样做，更复杂的事务处理可能需要部分提交或回退。

例如，前面描述的添加订单的过程为一个事务处理。如果发生错误，只需要返回到添加**orders**行之前即可，不需要回退到**customers**表(如果存在的话)。

为了支持回退部分事务处理，必须能在事务处理块中合适的位置放置占位符。这样，如果需要回退，可以回退到某个占位符。

这些占位符称为保留点。为了创建占位符，可如下使用**SAVEPOINT** 语句：

```
SAVEPOINT delete1;
#每个保留点都取标识它的唯一名字，以便在回退时，MySQL知道要回退到何处。为了回退到本例给出的保留点
ROLLBACK TO delete1;
```

释放保留点 保留点在事务处理完成(执行一条**ROLLBACK**或 **COMMIT**)后自动释放。自**MySQL 5**以来，也可以用**RELEASE SAVEPOINT**明确地释放保留点。

更改默认的提交行为

默认的MySQL行为是自动提交所有更改。

换句话说，任何时候你执行一条MySQL语句，该语句实际上都是针对表执行的，而且所做的更改立即生效。

为指示MySQL不自动提交更改，需要使用以下语句

```
SET autocommit=0;
```

autocommit标志决定是否自动提交更改，不管有没有COMMIT语句。

设置autocommit为0(假)指示MySQL不自动提交更改 (直到autocommit被设置为真为止)。

标志为连接专用 autocommit标志是针对每个连接而不是服务器的。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间： 2020-03-23 23:05:29

- MySQL用户管理
 - 查询所有用户
 - 创建用户账号
 - 重命名用户账号
 - 删除用户账号
 - 设置访问权限
 - 撤销权限
 - 权限
 - 更改口令

MySQL用户管理

查询所有用户

MySQL用户账号和信息存储在名为mysql的MySQL数据库中。一般不需要直接访问mysql数据库和表，但有时需要直接访问。需要直接访问它的时机之一是在需要获得所有用户账号列表时。

```
USE MYSQL;  
SELECT user FROM user;  
# (列名是user 表明也是user)
```

输出

+	-	-	-	-	+
		user			
+	-	-	-	-	+
		root			
+	-	-	-	-	+

mysql数据库有一个名为user的表，它包含所有用户账号。

user表有一个名为user的列，它存储用户登录名。

新安装的服务器可能只有一个用户(如这里所示)，过去建立的服务器可能具有很多用户。

创建用户账号

为了创建一个新用户账号，使用CREATE USER语句，如下所示：

```
CREATE USER ben IDENTIFIED BY 'p@$wOrd'
```

CREATE USER创建一个新用户账号。在创建用户账号时不一定需要口令，不过这个例子用IDENTIFIED BY 'p@\$wOrd'给出了一个口令。

指定散列口令

IDENTIFIED BY指定的口令为纯文本，MySQL 将在保存到user表之前对其进行加密。为了作为散列值指定口令，使用IDENTIFIED BY PASSWORD。

使用GRANT或INSERT

GRANT语句也可以创建用户账号，但一般来说CREATE USER是最清楚和最简单的句子。

此外，也可以通过直接插入行到user表来增加用户，不过为安全起见，一般不建议这样做。

重命名用户账号

为重新命名一个用户账号，使用RENAME USER语句，如下所示：

```
RENAME USER ben TO tina;
```

删除用户账号

为了删除一个用户账号(以及相关的权限)，使用DROP USER语句

```
DROP USER ben;
```

设置访问权限

在创建用户账号后，必须接着分配访问权限。新创建的用户账号没有访问权限。它们能登录MySQL，但不能看到数据，不能执行任何数据库操作。

为看到赋予用户账号的权限，使用SHOW GRANTS FOR

```
SHOW GRANTS FOR ben;
```

输出

```
+-----+
| Grants for bforta@% |
+-----+
| GRANT USAGE ON *.* TO 'bforta'@'%' |
+-----+
```

输出结果显示用户bforta有一个权限USAGE ON *.*。

USAGE表示根本没有权限，所以，此结果表示在任意数据库和任意表上对任何东西没有权限。

用户定义为**user@host** MySQL的权限用用户名和主机名结合定义。如果不指定主机名，则使用默认的主机名%(授予用户访问权限而不管主机名)。

为设置权限，使用**GRANT**语句。**GRANT**要求你至少给出以下信息：

- 要授予的权限；
- 被授予访问权限的数据库或表；
- 用户名。

以下例子给出GRANT的用法：

```
GRANT SELECT ON abc(数据库名字).* TO ben;
```

此GRANT允许用户在 abc.*(abc数据库上的所有表)上使用SELECT。

通过只授予SELECT访问权限，用户ben 对abc数据库中的所有数据具有只读访问权限。

SHOW GRANTS反映这个更改：

```
SHOW GRANTS FOR ben;
```

输出

```
+-----+
| Grants for bforta@% |
+-----+
| GRANT USAGE ON *.* TO 'bforta'@'%' |
| GRANT SELECT ON 'crashcourse'.* TO 'bforta'@'%' |
+-----+
```

每个GRANT添加(或更新)用户的一个权限。MySQL读取所有授权，并根据它们确定权限。

撤销权限

GRANT的反操作为**REVOKE**，用它来撤销特定的权限。下面举一个例子：

```
REVOKE SELECT ON abc(数据库名字).* TO ben;
```

这条REVOKE语句取消刚赋予用户ben的SELECT访问权限。被撤销的访问权限必须存在，否则会出错。

权限

GRANT和**REVOKE**可在几个层次上控制访问权限：

- 整个服务器，使用GRANT ALL和REVOKE ALL; □ 整个数据库，使用ON database.*;
- 特定的表，使用ON database.table;
- 特定的列;
- 特定的存储过程。

权限表

权限	说明
ALL	除GRANT OPTION外的所有权限
ALTER	使用ALTER TABLE
ALTER ROUTINE	使用ALTER PROCEDURE和DROP PROCEDURE
CREATE	使用CREATE TABLE
CREATE ROUTINE	使用CREATE PROCEDURE
CREATE TEMPORARY TABLES	使用CREATE TEMPORARY TABLE
CREATE USER	使用CREATE USER、DROP USER、RENAME USER和REVOKE ALL PRIVILEGES
CREATE VIEW	使用CREATE VIEW
DELETE	使用DELETE
DROP	使用DROP TABLE
EXECUTE	使用CALL和存储过程
FILE	使用SELECT INTO OUTFILE和LOAD DATA INFILE
GRANT OPTION	使用GRANT和REVOKE
INDEX	使用CREATE INDEX和DROP INDEX
INSERT	使用INSERT
LOCK TABLES	使用LOCK TABLES
PROCESS	使用SHOW FULL PROCESSLIST
RELOAD	使用FLUSH
REPLICATION CLIENT	服务器位置的访问
REPLICATION SLAVE	由复制从属使用
SELECT	使用SELECT
SHOW DATABASES	使用SHOW DATABASES
SHOW VIEW	使用SHOW CREATE VIEW
SHUTDOWN	使用mysqladmin shutdown(用来关闭MySQL)
SUPER	使用CHANGE MASTER、KILL、LOGS、PURGE、MASTER和SET GLOBAL。还允许mysqladmin调试登录

权限	说明
UPDATE	使用UPDATE
USAGE	无访问权限

使用GRANT和REVOKE，再结合表中列出的权限，你能对用户可以对你的宝贵数据做什么事情和不能做什么事情具有完全的控制。

简化多次授权,可通过列出各权限并用逗号分隔，将多条 GRANT语句串在一起，如下所示：

```
GRANT SELECT,INSERT ON DBName.* TO ben;
```

更改口令

为了更改用户口令，可使用SET PASSWORD语句。新口令必须如下加密：

```
SET PASSWORD FOR ben = Password ('new Password');
```

SET PASSWORD更新用户口令。新口令必须传递到Password()函数进行加密。

SET PASSWORD还可以用来设置你自己的口令：

```
SET PASSWORD = Password ('new Password');
```

在不指定用户名时，SET PASSWORD更新当前登录用户的口令。

Copyright © dujiaju.net 2020 all right reserved, powered by Gitbook该文件修订时间： 2020-03-27 00:46:54