MySQL 8.0参考手册

抽象

这是MySQL™参考手册。它记录了MySQL 8.0到8.0.15。它可能包括尚未发布的MySQL版本功能的文档。有关已发布的版本的信息，请参阅 MySQL 8.0发行说明。

MySQL 8.0目前不支持MySQL Cluster。有关MySQL Cluster的信息，请参阅MySQL NDB Cluster 7.5和NDB Cluster 7.6。

MySQL 8.0的功能。 本手册介绍了每个版本的MySQL 8.0中未包含的功能; 此类功能可能不包含在许可给您的MySQL 8.0版本中。如果您对MySQL 8.0版本中包含的功能有任何疑问，请参阅MySQL 8.0许可协议或联系您的Oracle销售代表。

有关详细说明每个版本中的更改的说明，请参阅 MySQL 8.0发行说明。

有关法律信息（包括许可信息），请参阅 前言和法律声明。

有关使用MySQL的帮助，请访问 MySQL论坛或 MySQL邮件列表，在那里您可以与其他MySQL用户讨论您的问题。

文件生成于：2018-10-11（修订：59436）

前言和法律声明

这是MySQL数据库系统8.0版到8.0.15版的参考手册。MySQL 8.0的次要版本之间的差异在本文中引用了版本号（8.0。 *x* ）。有关许可证信息，请参阅法律声明。

由于MySQL 8.0和以前版本之间存在许多功能和其他差异，因此本手册不适用于旧版本的MySQL软件。如果您使用的是早期版本的MySQL软件，请参阅相应的手册。例如， *MySQL 5.7参考手册* 涵盖5.7系列MySQL软件版本。

许可信息 - MySQL 8.0。 本产品可能包含经许可使用的第三方软件。如果您使用的是MySQL 8.0 的*商业*版本，请参阅 MySQL 8.0商业版许可证信息用户手册以获取许可信息，包括可能包含在此商业版本中的与第三方软件相关的许可信息。如果您使用的是MySQL 8.0 的 *社区*版本，请参阅 MySQL 8.0社区版本许可信息用户手册以获取许可信息，包括可能包含在此社区版本中的与第三方软件相关的许可信息。

法律声明

Micro Devices的商标或注册商标。UNIX是The Open Group的注册商标。

该软件或硬件和文档可以提供对来自第三方的内容，产品和服务的访问或信息。除非您与Oracle之间的适用协议另有规定，否则Oracle Corporation及其附属公司不对第三方内容，产品和服务的任何形式的保证承担任何责任，并且明确拒绝承担任何形式的保证。Oracle Corporation及其附属公司对由于您访问或使用第三方内容，产品或服务而导致的任何损失，成本或损害不承担任何责任，但您与Oracle之间的适用协议中规定的除外。

本文档不是根据GPL许可证分发的。使用本文档须遵守以下条款：

您可以仅为个人使用创建本文档的打印副本。只要不以任何方式更改或编辑实际内容，就允许转换为其他格式。您不得以任何形式或在任何媒体上发布或分发本文档，除非您以类似于Oracle传播文档的方式（即通过电子方式在网站上下载软件）或CD上发布文档。 -ROM或类似介质，但前提是文档与软件一起在同一介质上传播。任何其他用途，例如在另一份出版物中全部或部分传播打印副本或使用本文档，都需要得到Oracle授权代表的事先书面许可。

文档可访问性

有关Oracle对可访问性的承诺的信息，请访问Oracle Accessibility Program网站 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc。

访问Oracle支持

已购买支持的Oracle客户可通过My Oracle Support获得电子支持。有关详细信息，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info 或访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs 如果您听力受损。

第1章一般信息

目录

MySQL™软件提供了一个非常快速，多线程，多用户，强大的SQL（结构化查询语言）数据库服务器。MySQL Server适用于关键任务，重载生产系统以及嵌入大规模部署的软件。Oracle是Oracle Corporation和/或其附属公司的注册商标。MySQL是Oracle Corporation和/或其附属公司的商标，未经Oracle明确书面授权，客户不得使用。其他名称可能是其各自所有者的商标。

MySQL软件是Dual Licensed。用户可以选择根据GNU通用公共许可证（http://www.fsf.org/licenses/）的条款将MySQL软件用作开源产品，也可以从Oracle购买标准商业许可证。有关我们的许可政策的更多信息，请 访问http://www.mysql.com/company/legal/licensing/。

以下列表描述了本手册中特别感兴趣的部分内容：

- 有关MySQL数据库服务器功能的讨论，请参见 第1.3.2节“MySQL的主要功能”。
- 有关新MySQL功能的概述，请参见第1.4节“MySQL 8.0中的新增功能 ”。有关每个版本中的更改的信息，请参阅 发行说明。
- 有关安装说明，请参阅第2章，*安装和升级MySQL*。有关升级MySQL的信息，请参见 第2.11.1节“升级MySQL”。
- 有关MySQL数据库服务器的教程简介，请参阅 第3章*教程*。
- 有关配置和管理MySQL Server的信息，请参阅第5章，*MySQL服务器管理*。
- 有关MySQL安全性的信息，请参阅 第6章，*安全性*。
- 有关设置复制服务器的信息，请参见 第17章，*复制*。
- 有关MySQL Enterprise（具有高级功能和管理工具的商业MySQL版本）的信息，请参阅 第29章，*MySQL Enterprise Edition*。

- 有关MySQL数据库服务器及其功能的*常见问题的*答案，请参阅 附录A，*MySQL 8.0常见问题解答*。
- 有关新功能和错误修复的历史记录，请参阅 发行说明。

> **重要**
>
> 要报告问题或错误，请使用 第1.7节“如何报告错误或问题”中的说明。如果您在MySQL服务器中发现敏感的安全漏洞，请立即通过发送电子邮件告知我们。例外：支持客户应向Oracle支持部门报告所有问题，包括安全漏洞。 `<secalert_us@oracle.com>`

1.1关于本手册

这是MySQL数据库系统8.0版到8.0.15版的参考手册。MySQL 8.0的次要版本之间的差异在本文中引用了版本号（8.0。*x*）。有关许可证信息，请参阅法律声明。

由于MySQL 8.0和以前版本之间存在许多功能和其他差异，因此本手册不适用于旧版本的MySQL软件。如果您使用的是早期版本的MySQL软件，请参阅相应的手册。例如，*MySQL 5.7参考手册*涵盖5.7系列MySQL软件版本。

由于本手册可作为参考，因此不提供有关SQL或关系数据库概念的一般说明。它也没有教你如何使用操作系统或命令行解释器。

MySQL数据库软件正在不断发展，参考手册也经常更新。最新版本的手册可在线搜索，网址为https://dev.mysql.com/doc/。其他格式也可用，包括HTML，PDF和EPUB版本。

参考手册源文件以DocBook XML格式编写。HTML版本和其他格式是自动生成的，主要使用DocBook XSL样式表。有关DocBook的信息，请参阅http://docbook.org/

MySQL本身的源代码包含使用Doxygen编写的内部文档。生成的Doxygen内容可从https://dev.mysql.com/doc/dev/mysql-server/latest/获得。也可以使用第2.9.7节“生成MySQL Doxygen文档内容”中的说明从MySQL源代码分发本地生成此内容 。

如果您对使用MySQL有疑问，可以使用我们的邮件列表或论坛询问他们。请参见第1.6.2节“MySQL邮件列表”和第1.6.3节“MySQL论坛中的MySQL社区支持”。如果您有关于手册本身的添加或更正的建议，请将它们发送到http://www.mysql.com/company/contact/。

本手册最初由David Axmark和Michael “ Monty ” Widenius 编写 。它由MySQL文档团队维护，包括Chris Cole，Paul DuBois，Margaret Fisher，Edward Gilmore，Stefan Hinz，David Moss，Philip Olson，Daniel Price，Daniel So和Jon Stephens。

1.2印刷和语法约定

本手册使用某些印刷约定：

- `Text in this style`用于SQL语句; 数据库，表和列名称; 程序清单和源代码; 和环境变量。示例：“ 要重新加载授权表，请使用该`FLUSH PRIVILEGES`语句。”
- **`Text in this style`** 表示您在示例中键入的输入。
- **此样式的文本**表示可执行程序和脚本的名称，例如 **mysql**（MySQL命令行客户端程序）和 **mysqld**（MySQL服务器可执行文件）。
- ***`Text in this style`*** 用于变量输入，您应该替换自己选择的值。
- *此样式的文本*用于强调。
- ***这种风格的文字***用于表格标题，并特别强调。
- `Text in this style`用于指示程序选项，该程序选项影响程序的执行方式，或者提供程序以某种方式运行所需的信息。*例如*：“ 该`--host`选项（缩写形式`-h`）告诉**MySQL的**客户端程序，它应该连接到MySQL服务器的主机名或IP地址 ”。
- 文件名和目录名的写法如下：“ 全局`my.cnf`文件位于`/etc`目录中。”
- 字符序列的写法如下：“ 要指定通配符，请使用' `%`' 字符。”

当显示要在特定程序内执行的命令时，命令前面显示的提示指示要使用的命令。例如，`shell>` 表示您从登录shell执行的命令 `root-shell>`类似但应该执行`root`，并`mysql>` 指示您从**mysql**客户端程序执行的语句 ：

```
shell> type a shell command here
root-shell> type a shell command as root here
mysql> type a mysql statement here
```

在一些区域中，可以将不同的系统彼此区分开以示出命令应该在两个不同的环境中执行。例如，复制工作时的命令可能与前缀`master`和 `slave`：

```
master> type a mysql command on the replication master here
slave> type a mysql command on the replication slave here
```

该“ 壳 ”是您的命令解释器。在Unix上，这通常是一个程序，如**sh**， **csh**或**bash**。在Windows上，等效程序是**command.com**或 **cmd.exe**，通常在控制台窗口中运行。

输入示例中显示的命令或语句时，请不要键入示例中显示的提示。

数据库，表和列名称通常必须替换为语句。为了指示这样的取代是必要的，本手册使用***db_name***，***tbl_name***和 ***col_name***。例如，您可能会看到如下语句：

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

这意味着如果您要输入类似的语句，您将提供自己的数据库，表和列名称，可能如下所示：

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL关键字不区分大小写，可以用任何字母大小写。本手册使用大写。

在语法描述中，方括号（“[”和“]”）表示可选的单词或子句。例如，在以下语句中，IF EXISTS是可选的：

```
DROP TABLE [IF EXISTS] tbl_name
```

当语法元素由许多替代项组成时，替代项由竖线（“|”）分隔。当从一组的一个选择部件*可以*被选择，则将替换方括号内所列的（“[”和“]”）：

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

当*必须*选择一组选项中的一个成员时，替代项将在大括号（“{”和“}”）中列出：

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

省略号（...）表示省略语句的一部分，通常是为了提供更复杂语法的更短版本。例如，SELECT ... INTO OUTFILE是SELECT语句形式的简写，该 语句INTO OUTFILE在语句的其他部分后面有一个 子句。

省略号还可以指示可以重复语句的前述语法元素。在下面的示例中，***reset_option***可以给出多个 值，每个值在第一个之后以逗号开头：

```
RESET reset_option [,reset_option] ...
```

使用Bourne shell语法显示用于设置shell变量的命令。例如，设置CC 环境变量并运行**configure** 命令的顺序在Bourne shell语法中如下所示：

```
shell> CC=gcc ./configure
```

如果您使用的是**csh**或**tcsh**，则必须以不同的方式发出命令：

```
shell> setenv CC gcc
shell> ./configure
```

1.3 MySQL数据库管理系统概述

1.3.1什么是MySQL？

MySQL是最受欢迎的开源SQL数据库管理系统，由Oracle Corporation开发，分发和支持。

MySQL网站（http://www.mysql.com/）提供有关MySQL软件的最新信息。

- ***MySQL是一个数据库管理系统。***

  数据库是数据的结构化集合。它可以是从简单的购物清单到图片库或公司网络中的大量信息。要添加，访问和处理存储在计算机数据库中的数据，您需要一个数据库管理系统，如MySQL Server。由于计算机非常擅长处理大量数据，因此数据库管理系统在计算，独立实用程序或其他应用程序的一部分中起着核心作用。

- ***MySQL数据库是关系型的。***

  关系数据库将数据存储在单独的表中，而不是将所有数据放在一个大的库房中。数据库结构被组织成针对速度优化的物理文件。逻辑模型具有数据库，表，视图，行和列等对象，可提供灵活的编程环境。您可以设置管理不同数据字段之间关系的规则，例如一对一，一对多，唯一，必需或可选，以及 不同表之间的“ 指针 ”。数据库强制执行这些规则，因此使用设计良好的数据库，您的应用程序永远不会看到不一致，重复，孤立，过时或丢失的数据。

  “ MySQL ”的SQL部分代表“ 结构化查询语言 ”。SQL是用于访问数据库的最常用的标准化语言。根据您的编程环境，您可以直接输入SQL（例如，生成报告），将SQL语句嵌入到用其他语言编写的代码中，或使用隐藏SQL语法的特定于语言的API。

  SQL由ANSI / ISO SQL标准定义。SQL标准自1986年以来一直在发展，并且存在多个版本。在本手册中，“ SQL-92 ”是指1992年发布的标准，“ SQL：1999 ”是指1999年发布的标准，“ SQL：2003 ”是指当前版本的标准。我们在任何时候都使用短语“ SQL标准 ”来表示当前版本的SQL标准。

- ***MySQL软件是开源的。***

  开源意味着任何人都可以使用和修改软件。任何人都可以从互联网上下载MySQL软件并使用它而无需支付任何费用。如果您愿意，您可以学习源代码并根据您的需要进行更改。MySQL软件使用GPL（GNU通用公共许可证）http://www.fsf.org/licenses/来定义在不同情况下您可能和不可以使用该软件的内容。如果您对GPL感到不舒服或需要将MySQL代码嵌入到商业应用程序中，您可以从我们这里购买商业许可版本。有关更多信息，请参阅MySQL许可概述
  （http://www.mysql.com/company/legal/licensing/）。

- ***MySQL数据库服务器非常快速，可靠，可扩展且易于使用。***

  如果这是你正在寻找的，你应该试一试。MySQL服务器可以在台式机或笔记本电脑上与其他应用程序，Web服务器等一起轻松运行，几乎不需要关注。如果您将整台机器专用于MySQL，则可以调整设置以利用所有可用的内存，CPU功率和I / O容量。MySQL还可以扩展到联网的机器集群。

MySQL Server最初开发用于处理大型数据库的速度比现有解决方案快得多，并且已成功应用于高要求的生产环境中数年。虽然在不断发展的今天，MySQL Server提供了丰富而有用的功能集。它的连接性，速度和安全性使MySQL Server非常适合访问Internet上的数据库。

- ***MySQL Server适用于客户端/服务器或嵌入式系统。***

  MySQL数据库软件是一个客户端/服务器系统，由支持不同后端的多线程SQL服务器，几个不同的客户端程序和库，管理工具以及各种应用程序编程接口（API）组成。

  我们还提供MySQL Server作为嵌入式多线程库，您可以将其链接到您的应用程序，以获得更小，更快，更易于管理的独立产品。

- ***提供了大量贡献的MySQL软件。***

  MySQL Server具有与我们的用户密切合作开发的一组实用功能。您最喜欢的应用程序或语言很可能支持MySQL数据库服务器。

发音为“ MySQL ”的官方方式是“ My Ess Que Ell ”（不是“ 我的续集 ”），但我们不介意你把它发音为“ 我的续集 ”还是用其他一些本地化的方式。

1.3.2 MySQL的主要特点

本节介绍MySQL数据库软件的一些重要特性。在大多数方面，该路线图适用于所有版本的MySQL。有关在特定于系列的基础上引入MySQL的功能的信息，请参阅相应手册的 “ 在坚果壳中 ”部分：

- MySQL 8.0：第1.4节“MySQL 8.0中的新功能”
- MySQL 5.7：MySQL 5.7中的新功能
- MySQL 5.6：MySQL 5.6中的新功能
- MySQL 5.5：MySQL 5.5中的新功能

内部和可移植性

- 用C和C ++编写。
- 经过广泛的不同编译器测试。
- 适用于许多不同的平台。请参阅 https://www.mysql.com/support/supportedplatforms/database.html。
- 为了便于携带，在MySQL 5.5及更高版本中使用**CMake**。以前的系列使用GNU Automake，Autoconf和Libtool。
- 使用Purify（商业内存泄漏检测器）以及GPL工具Valgrind（http://developer.kde.org/~sewardj/）进行测试。
- 使用具有独立模块的多层服务器设计。
- 设计为使用内核线程完全多线程，以便在可用时轻松使用多个CPU。
- 提供事务性和非事务性存储引擎。
- 使用`MyISAM`具有索引压缩的非常快速的B树磁盘表（）。
- 旨在使添加其他存储引擎相对容易。如果要为内部数据库提供SQL接口，这非常有用。
- 使用非常快速的基于线程的内存分配系统。
- 使用优化的嵌套循环连接执行非常快速的连接。
- 实现内存中的哈希表，用作临时表。
- 使用应尽可能快的高度优化的类库实现SQL函数。通常在查询初始化之后根本没有内存分配。

- 将服务器作为单独的程序提供，用于客户端/服务器网络环境，以及作为可嵌入（链接）到独立应用程序的库。此类应用程序可以单独使用，也可以在没有网络的环境中使用。

数据类型

- 许多数据类型：有符号/无符号整数1，2，3，4，和8个字节长，FLOAT，DOUBLE，CHAR，VARCHAR，BINARY，VARBINARY，TEXT，BLOB，DATE，TIME，DATETIME，TIMESTAMP，YEAR，SET，ENUM，和开放GIS空间类型。请参见第11章，*数据类型*。

- 固定长度和可变长度的字符串类型。

陈述和职能

- 查询SELECT列表和 WHERE子句中的 完整运算符和函数支持 。例如：

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
    -> FROM citizen
    -> WHERE income/dependents > 10000 AND age > 30;
```

- 完全支持SQL GROUP BY和 ORDER BY子句。支持基函数（COUNT()，AVG()，STD()，SUM()，MAX()，MIN()，和 GROUP_CONCAT()）。

- 支持LEFT OUTER JOIN和 RIGHT OUTER JOIN使用标准SQL和ODBC语法。

- 根据标准SQL的要求支持表和列上的别名。

- 支持DELETE，INSERT，REPLACE，和 UPDATE以返回更改（受影响）的行数，或返回通过连接到服务器时设置标志，而不是匹配的行的数量。

- 支持特定于MySQL的SHOW 语句，用于检索有关数据库，存储引擎，表和索引的信息。支持 INFORMATION_SCHEMA数据库，根据标准SQL实现。

- 一个EXPLAIN语句来显示优化器如何解决一个查询。

- 表名或列名中函数名的独立性。例如，ABS是一个有效的列名。唯一的限制是，对于函数调用，函数名和它后面的“(”之间不允许有空格 。请参见 第9.3节“关键字和保留字”。

- 您可以在同一语句中引用来自不同数据库的表。

安全

- 特权和密码系统，非常灵活和安全，可以进行基于主机的验证。

- 连接到服务器时加密所有密码流量的密码安全性。

可扩展性和限制

- 支持大型数据库。我们将MySQL Server与包含5000万条记录的数据库结合使用。我们也知道使用MySQL服务器的用户有200,000个表和大约5,000,000,000行。

- 每个表最多支持64个索引。每个索引可以包含1到16列或部分列。InnoDB表的最大索引宽度为767字节或3072字节。请参见 第15.8.1.7节“InnoDB表的限制”。MyISAM表的最大索引宽度为1000个字节。请参见第16.2节“MyISAM存储引擎”。索引可使用的柱的前缀CHAR， VARCHAR， BLOB，或 TEXT列类型。

连接

- 客户端可以使用多种协议连接到MySQL Server：

  - 客户端可以在任何平台上使用TCP / IP套接字进行连接。

  - 在Windows系统上，如果使用该--enable-named-pipe选项启动服务器，则客户端可以使用命名管道进行连接 。如果使用该--shared-memory选项启动，Windows服务器也支持共享内存连接 。客户端可以使用该--protocol=memory选项通过共享内存进行连接 。

  - 在Unix系统上，客户端可以使用Unix域套接字文件进行连接。

- MySQL客户端程序可以用多种语言编写。用C编写的客户端库可用于使用C或C ++编写的客户端，也可用于提供C绑定的任何语言。

- 可以使用C，C ++，Eiffel，Java，Perl，PHP，Python，Ruby和Tcl的API，从而使MySQL客户端能够以多种语言编写。请参见第27章，*连接器和API*。

- Connector / ODBC（MyODBC）接口为使用ODBC（开放式数据库连接）连接的客户端程序提供MySQL支持。例如，您可以使用MS Access连接到MySQL服务器。客户端可以在Windows或Unix上运行。连接器/ ODBC源可用。与许多其他功能一样，支持所有ODBC 2.5功能。请参阅 MySQL Connector / ODBC开发人员指南。

- Connector / J接口为使用JDBC连接的Java客户端程序提供MySQL支持。客户端可以在Windows或Unix上运行。连接器/ J源可用。请参阅 MySQL Connector / J 5.1开发人员指南。

- MySQL Connector / NET使开发人员能够轻松创建需要与MySQL进行安全，高性能数据连接的.NET应用程序。它实现了所需的ADO.NET接口，并集成到ADO.NET感知工具中。开发人员可以使用他们选择的.NET语言构建应用程序。MySQL Connector / NET是一个完全托管的ADO.NET驱动程序，用100％纯C＃编写。请参阅 MySQL Connector / NET Developer Guide。

本土化

- 服务器可以以多种语言向客户端提供错误消息。请参见第10.11节“设置错误消息语言”。

- 几个不同的字符集，包括全面支持 latin1（CP1252），german， big5，ujis一些Unicode字符集等。例如，表格和列名称中允许使用斯堪的纳维亚字符“å”，“ä”和“ö”。

- 所有数据都保存在所选字符集中。

- 根据默认字符集和排序规则进行排序和比较。启动MySQL服务器时可以更改此项（请参见 第10.3.2节“服务器字符集和排序”）。要查看非常高级排序的示例，请查看捷克排序代码。MySQL Server支持许多可在编译时和运行时指定的不同字符集。
- 可以动态更改服务器时区，并且各个客户端可以指定自己的时区。请参见 第5.1.12节“MySQL服务器时区支持”。

客户端和工具

- MySQL包括几个客户端和实用程序。这些包括命令行程序，如 **mysqldump**和 **mysqladmin**，以及图形程序，如 MySQL Workbench。
- MySQL Server内置支持SQL语句来检查，优化和修复表。这些语句可以从命令行通过 **mysqlcheck**客户端获得。MySQL还包括 **myisamchk**，这是一个非常快速的命令行实用程序，用于在`MyISAM` 表上执行这些操作。请参见第4章，*MySQL程序*。
- 可以使用`--help` 或`-?`选项调用MySQL程序以获取在线帮助。

1.3.3 MySQL的历史

我们最初的目的是使用 mSQL数据库系统使用我们自己的快速低级（ISAM）例程连接到我们的表。但是，经过一些测试，我们得出的结论mSQL 是，对于我们的需求来说，这个结论还不够快或不够灵活。这导致了一个新的SQL接口到我们的数据库，但具有几乎相同的API接口mSQL。此API旨在使用于编写的第三方代码能够mSQL轻松移植以与MySQL一起使用。

MySQL以联合创始人Monty Widenius的女儿My的名字命名。

MySQL Dolphin（我们的标识）的名称是“ Sakila ”，它是从我们的“ 海豚名人”比赛用户建议的大量名单中选出的。获奖名称由非洲斯威士兰的开源软件开发商Ambrose Twebaze提交。根据安布罗斯的说法，女性名称Sakila的根源在于斯威士兰当地语言SiSwati。Sakila也是坦桑尼亚阿鲁沙一个小镇的名字，靠近Ambrose的原籍乌干达。

1.4 MySQL 8.0中有哪些新功能

本节总结了MySQL 8.0中添加，弃用和删除的内容。随附部分列出了在MySQL 8.0中添加，弃用或删除的MySQL服务器选项和变量。请参见 第1.5节“在MySQL 8.0中添加，弃用或删除的服务器和状态变量和选项”。

- MySQL 8.0中添加的功能
- 功能在MySQL 8.0中不推荐使用
- MySQL 8.0中删除的功能

MySQL 8.0中添加的功能

MySQL 8.0中添加了以下功能：

- **数据字典。** MySQL现在包含一个事务数据字典，用于存储有关数据库对象的信息。在以前的MySQL版本中，字典数据存储在元数据文件和非事务表中。有关更多信息，请参见第14章，*MySQL数据字典*。
- **原子数据定义语句（Atomic DDL）。** 原子DDL语句将与DDL操作关联的数据字典更新，存储引擎操作和二进制日志写入组合到单个原子事务中。有关更多信息，请参见 第13.1.1节“原子数据定义语句支持”。
- **安全和帐户管理。** 添加了这些增强功能以提高安全性并在帐户管理中实现更高的DBA灵活性：
  - `mysql`系统数据库中 的授权表现在是`InnoDB`（事务性）表。以前，这些是 `MyISAM`（非交易）表。授权表存储引擎的更改是对帐户管理语句行为的伴随变更的基础。以前，一个帐户管理声明（如 `CREATE USER`或 `DROP USER`）命名多个用户可以为某些用户成功而对其他用户失败。现在，每个语句都是事务性的，并且对所有命名用户都成功或回滚，如果发生任何错误则无效。如果成功，则将语句写入二进制日志，但如果失败则不写入; 在这种情况下，发生回滚并且不进行任何更改。有关更多信息，请参见第13.1.1节“原子数据定义语句支持”。
  - 可以使用新的`caching_sha2_password` 身份验证插件。与`sha256_password`插件一样 ，`caching_sha2_password`实现SHA-256密码散列，但使用缓存来解决连接时的延迟问题。它还支持更多连接协议，并且不需要针对基于RSA密钥对的密码交换功能与OpenSSL链接。请参见 第6.5.1.3节“缓存SHA-2可插入认证”。

    该`caching_sha2_password`和 `sha256_password`认证插件提供比更安全的密码加密 `mysql_native_password`插件，并 `caching_sha2_password`提供了比更好的性能`sha256_password`。由于这些优越的安全性和性能特性 `caching_sha2_password`，它现在是首选的身份验证插件，而且也是默认的身份验证插件而不是 `mysql_native_password`。有关此默认插件更改对服务器操作的影响以及服务器与客户端和连接器的兼容性的信息，请参阅 caching_sha2_password作为首选身份验证插件。
  - MySQL现在支持角色，这些角色被命名为特权集合。可以创建和删除角色。角色可以拥有授予和撤消的权限。可以从用户帐户授予和撤消角色。可以从授予帐户的帐户中选择帐户的活动适用角色，并且可以在该帐户的会话期间更改。有关更多信息，请参见第6.3.4节“使用角色”。
  - MySQL现在维护有关密码历史的信息，从而限制了以前密码的重用。DBA可能要求在某些密码更改或时间段内不从先前的密码中选择新密码。可以在全局以及每个帐户的基础上建立密码重用策略。

    还可以要求通过指定要替换的当前密码来验证更改帐户密码的尝试。这使DBA能够阻止用户更改密码，而无需证明他们知道当前密码。可以在全球以及每个帐户的基础上建立密码验证策略。

    这些新功能为DBA提供了更完整的密码管理控制。有关更多信息，请参见第6.3.8节“密码管理”。

- MySQL现在支持FIPS模式，如果使用OpenSSL编译，并且OpenSSL库和FIPS对象模块在运行时可用。FIPS模式对加密操作施加了条件，例如对可接受的加密算法的限制或对更长密钥长度的要求。请参见第6.6节“FIPS支持”。

- **资源管理。** MySQL现在支持资源组的创建和管理，并允许将服务器内运行的线程分配给特定组，以便线程根据组可用的资源执行。组属性可以控制其资源，以启用或限制组中线程的资源消耗。DBA可以根据不同的工作负载修改这些属性。目前，CPU时间是一种可管理的资源，以“ 虚拟CPU ”的概念为代表作为包含CPU核心，超线程，硬件线程等的术语。服务器在启动时确定可用的虚拟CPU数量，具有适当权限的数据库管理员可以将这些CPU与资源组关联，并将线程分配给组。有关更多信息，请参见 第8.12.5节“资源组”。

- **InnoDB增强功能。** 增加了这些InnoDB增强功能：

  - 每次值更改时，当前最大自动增量计数器值将写入重做日志，并保存到每个检查点上的引擎专用系统表中。这些更改使当前最大自动增量计数器值在服务器重新启动时保持不变。另外：

    - 服务器重新启动不再取消AUTO_INCREMENT = N表选项的效果 。如果将自动递增计数器初始化为特定值，或者将自动递增计数器值更改为更大的值，则新值将在服务器重新启动时保持不变。

    - 在ROLLBACK 操作之后立即重新启动服务器 不再导致重用已分配给回滚事务的自动增量值。

    - 如果将AUTO_INCREMENT 列值修改为大于当前最大自动增量值的值（ UPDATE例如，在操作中），则新值将保持不变，后续 INSERT操作将从新的较大值开始分配自动增量值。

    有关更多信息，请参见 第15.8.1.5节“InnoDB中的AUTO_INCREMENT处理”和 InnoDB AUTO_INCREMENT计数器初始化。

  - 遇到索引树损坏时， InnoDB将损坏标志写入重做日志，这会使损坏标志崩溃安全。InnoDB还将内存中损坏标志数据写入每个检查点上的引擎专用系统表。在恢复期间， InnoDB从两个位置读取损坏标志并在将内存表和索引对象标记为损坏之前合并结果。

  - 的InnoDB **分布式缓存**插件支持多个 get操作（读取在一个单一的多键/值对**分布式缓存** 查询）和范围查询。请参见第15.19.4节“InnoDB memcached多个获取和范围查询支持”。

  - 新的动态配置选项 innodb_deadlock_detect可用于禁用死锁检测。在高并发系统上，当许多线程等待同一个锁时，死锁检测会导致速度减慢。有时，禁用死锁检测可能更有效，并且在innodb_lock_wait_timeout 发生死锁时依赖于事务回滚的 设置。

  - 新 INFORMATION_SCHEMA.INNODB_CACHED_INDEXES 表报告InnoDB每个索引缓冲池中缓存的索引页数 。

  - InnoDB现在，在共享临时表空间中创建临时表 ibtmp1。

  - 该InnoDB 表空间加密功能支持重做日志的加密和撤消日志数据。请参阅 重做日志数据加密和 撤消日志数据加密。

  - InnoDB支持 NOWAIT和SKIP LOCKED选项SELECT ... FOR SHARE以及SELECT ... FOR UPDATE锁定读取语句。NOWAIT如果请求的行被另一个事务锁定，则会立即返回该语句。SKIP LOCKED从结果集中删除锁定的行。请参阅 使用NOWAIT和SKIP LOCKED锁定读取并发。

    SELECT ... FOR SHARE替换 SELECT ... LOCK IN SHARE MODE，但 LOCK IN SHARE MODE仍可用于向后兼容。这些陈述是等同的。然而，FOR UPDATE和 FOR SHARE支持 NOWAIT，SKIP LOCKED和选项。请参见第13.2.10节“SELECT语法”。 OF ***tbl_name***

    OF ***tbl_name*** 将锁定查询应用于命名表。

  - ADD PARTITION，DROP PARTITION，COALESCE PARTITION，REORGANIZE PARTITION，和REBUILD PARTITION ALTER TABLE选项由本地分区就地API的支持，可能与使用 ALGORITHM={COPY|INPLACE}和 LOCK条

款。

DROP PARTITION与 ALGORITHM=INPLACE存储在该分区删除数据并丢弃分区。但是， DROP PARTITION使用 ALGORITHM=COPY或 old_alter_table=ON 重建分区表并尝试将数据从已删除的分区移动到具有兼容PARTITION ... VALUES 定义的另一个分区。将删除无法移动到其他分区的数据。

- 该InnoDB存储引擎现在使用MySQL的数据字典，而不是它自己的存储引擎特定的数据字典。有关数据字典的信息，请参阅 第14章，*MySQL数据字典*。
- mysql系统表和数据字典表现在在MySQL数据目录中InnoDB命名的单个表空间文件中 创建 mysql.ibd。以前，这些表是InnoDB在mysql数据库目录中的各个表空间文件中创建的。
- MySQL 8.0中引入了以下撤消表空间更改：
  - 现在可以使用innodb_undo_tablespaces 配置选项在运行时或重新启动服务器时修改撤消表空间的数量 。此更改允许在数据库增长时添加撤消表空间和回滚段。
  - innodb_undo_log_truncate 默认情况下启用。请参见 第15.7.9节“截断撤消表空间”。
  - 的 innodb_undo_tablespaces 默认值从0变更为2，这意味着回退段在两个单独的还原表，而不是创建InnoDB默认系统表空间。至少需要两个undo表空间才能截断undo日志。

    最小值 innodb_undo_tablespaces 为2，innodb_undo_tablespaces 不再允许设置 为0。最小值2确保始终在撤消表空间而不是系统表空间中创建回滚段。有关更多信息，请参见 第15.7.8节“配置撤消表空间”。
  - 用于撤消表空间文件的命名约定从更改 为 ，其中 是撤消空间编号。 undo***NNN***undo_***NNN NNN***
  - 的 innodb_rollback_segments 配置选项定义每撤消表回退段的数目。以前， innodb_rollback_segments 是一个全局设置，指定了MySQL实例的回滚段总数。此更改会增加可用于并发事务的回滚段数。更多回滚段增加了并发事务使用单独的回滚段进行撤消日志的可能性，从而减少了资源争用。
  - 该innodb_undo_logs 配置选项被删除。的 innodb_rollback_segments 配置选项执行相同的功能，并应被代替使用。
  - 该Innodb_available_undo_logs 状态变量被删除。可以使用检索每个表空间的可用回滚段的数量SHOW VARIABLES LIKE 'innodb_rollback_segments';
- 影响缓冲池预冲洗和刷新行为的配置选项的默认值已修改：
  - 该 innodb_max_dirty_pages_pct_lwm 默认值现在是10。0先前的默认值禁用缓冲池预冲洗。当缓冲池中脏页的百分比超过10%时，值10将启用预刷。启用预冲洗可提高性能一致性。
  - 将 innodb_max_dirty_pages_pct 默认值从75到90。增加 InnoDB尝试从缓冲池刷新数据，使脏页的百分比不超过这个值。增加的默认值允许缓冲池中较大百分比的脏页。
- 默认 innodb_autoinc_lock_mode 设置现在为2（交错）。交错锁定模式允许并行执行多行插入，从而提高了并发性和可伸缩性。新的 innodb_autoinc_lock_mode 默认设置反映了从基于语句的复制到基于行的复制的更改，作为MySQL 5.7中的默认复制类型。基于语句的复制需要连续的自动增量锁定模式（以前的默认值），以确保为给定的SQL语句序列以可预测且可重复的顺序分配自动增量值，而基于行的复制对于SQL语句的执行顺序。有关更多信息，请参阅 InnoDB AUTO_INCREMENT锁定模式。

  对于使用基于语句的复制的系统，新的 innodb_autoinc_lock_mode 默认设置可能会破坏依赖于顺序自动增量值的应用程序。要恢复以前的默认值，请设置 innodb_autoinc_lock_mode 为1。

- ALTER TABLESPACE ... RENAME TO语法 支持重命名通用表空间 。
- innodb_dedicated_server 默认情况下禁用 的新 配置选项可用于InnoDB根据服务器上检测到的内存量自动配置以下选项：
  - innodb_buffer_pool_size
  - innodb_log_file_size
  - innodb_flush_method

  此选项适用于在专用服务器上运行的MySQL服务器实例。有关更多信息，请参见 第15.6.13节“为专用MySQL服务器启用自动配置”。
- 新 INFORMATION_SCHEMA.INNODB_TABLESPACES_BRIEF 视图为表空间提供空间，名称，路径，标志和空间类型数据InnoDB。
- 与MySQL捆绑在一起 的zlib库版本从1.2.3版本升级到版本1.2.11。MySQL在zlib库的帮助下实现压缩。

  如果使用InnoDB压缩表，请参见第2.11.1.3节“MySQL 8.0中的更改”以了解相关的升级含义。
- 序列化字典信息（SDI）存在于InnoDB除全局临时表空间和撤消表空间文件之外的所有表空间文件中。SDI是表和表空间对象的序列化元数据。SDI数据的存在提供了元数据冗余。例如，如果数据字典变得不可用，则可以从表空间文件中提取字典对象元数据。使用**ibd2sdi**工具执行SDI提取。SDI数据以JSON格式存储。

  在表空间文件中包含SDI数据会增加表空间文件的大小。SDI记录需要单个索引页面，默认情况下大小为16KB。但是，SDI数据在存储时会进行压缩，以减少存储空间。
- 该InnoDB存储引擎现在支持原子DDL，这保证了DDL操作要么完全提交或回滚，即使服务器在操作时停止。有关更多信息，请参见第13.1.1节“原子数据定义语句支持”。
- 使用该innodb_directories 选项在服务器脱机时，可以将表空间文件移动或还原到新位置 。有关更多信息，请参见 第15.7.7节“在服务器脱机时移动表空间文件”。
- 实施了以下重做日志记录优化：
  - 用户线程现在可以并发写入日志缓冲区而无需同步写入。
  - 用户线程现在可以以宽松的顺序将脏页添加到刷新列表中。
  - 专用日志线程现在负责将日志缓冲区写入系统缓冲区，将系统缓冲区刷新到磁盘，通知用户线程有关写入和刷新的重做，维护宽松刷新列表顺序所需的延迟以及写入检查点。
  - 添加了系统变量，用于配置等待刷新重做的用户线程使用旋转延迟：
    - innodb_log_wait_for_flush_spin_hwm：定义最大平均日志刷新时间，超过该时间，用户线程在等待刷新的重做时不再旋转。
    - innodb_log_spin_cpu_abs_lwm：定义在等待刷新重做时用户线程不再旋转的最小CPU使用量。
    - innodb_log_spin_cpu_pct_hwm：定义在等待刷新的重做时用户线程不再旋转的最大CPU使用量。
  - 该 innodb_log_buffer_size 配置选项现在是动态的，在服务器运行时，其允许调整日志缓冲区的。

  有关更多信息，请参见 第8.5.4节“优化InnoDB重做日志记录”。

- 从MySQL 8.0.12开始，对大对象（LOB）数据的小更新支持撤消日志记录，这可以提高大小为100字节或更小的LOB更新的性能。以前，LOB更新的大小至少为一个LOB页面，对于可能只修改几个字节的更新而言，这不是最佳选择。此增强功能建立在MySQL 8.0.4中为LOB数据的部分更新添加的支持的基础上。
- 从MySQL 8.0.12开始，`ALGORITHM=INSTANT` 支持以下 `ALTER TABLE`操作：
  - 添加列。此功能也称为 “ 即时`ADD COLUMN` ”。限制适用。请参见 第15.12.1节“在线DDL操作”。
  - 添加或删除虚拟列。
  - 添加或删除列默认值。
  - 修改`ENUM`或 `SET`列的定义 。
  - 更改索引类型。
  - 重命名表。

  `ALGORITHM=INSTANT`仅 支持 修改数据字典中的元数据的操作。表上没有元数据锁，表数据不受影响，使操作瞬间完成。如果未明确指定，`ALGORITHM=INSTANT`则默认情况下由支持它的操作使用。如果 `ALGORITHM=INSTANT`已指定但不受支持，则操作会立即失败并显示错误。

  有关支持的操作的更多信息 `ALGORITHM=INSTANT`，请参见 第15.12.1节“在线DDL操作”。
- 从MySQL 8.0.13开始，`TempTable` 存储引擎支持二进制大对象（BLOB）类型列的存储。此增强功能可提高使用包含BLOB数据的临时表的查询的性能。以前，包含BLOB数据的临时表存储在由...定义的磁盘存储引擎中 `internal_tmp_disk_storage_engine`。有关更多信息，请参见 第8.4.4节“MySQL中的内部临时表使用”。
- 从MySQL 8.0.13开始，`InnoDB` 表空间加密功能支持通用表空间。以前，只能对每个表的文件表空间进行加密。为了支持一般的表空间的加密，`CREATE TABLESPACE`以及`ALTER TABLESPACE`语法扩展到包括 `ENCRYPTION`条款。

  该 `INFORMATION_SCHEMA.INNODB_TABLESPACES` 表现在包含一个`ENCRYPTION` 列，用于指示表空间是否已加密。

  在`stage/innodb/alter tablespace (encryption)`加入绩效模式阶段仪器允许普通表空间加密操作的监控。
- 要减小核心文件的大小，`innodb_buffer_pool_in_core_file` 可以禁用该 变量以防止将 `InnoDB`缓冲池页面写入核心文件。
- 从MySQL 8.0.13开始，由优化器创建的用户创建的临时表和内部临时表存储在会话临时表空间中，这些表空间从临时表空间池分配给会话。会话断开连接时，其临时表空间将被截断并释放回池中。在以前的版本中，临时表是在全局临时表空间（`ibtmp1`）中创建的，在临时表被删除后，它不会将磁盘空间返回给操作系统。

  该 `innodb_temp_tablespaces_dir` 变量定义创建会话临时表空间的位置。默认位置是 `#innodb_temp`数据目录中的目录。

  该 `INNODB_SESSION_TEMP_TABLESPACES` 表提供有关会话临时表空间的元数据。

  全局临时表空间（`ibtmp1`）现在存储用于对用户创建的临时表所做更改的回滚段。

- **字符集支持。** 默认字符集已从更改 `latin1`为`utf8mb4`。该`utf8mb4`字符集有几个新的排序规则，其中包括 `utf8mb4_ja_0900_as_cs`，提供对Unicode在MySQL中第一个日本特定语言的排序规则。有关更多信息，请参见 第10.10.1节“Unicode字符集”。
- **JSON增强功能。** 对MySQL的JSON功能进行了以下增强或添加：

- 添加了 `->>`（内联路径）运算符，相当于调用 `JSON_UNQUOTE()` 结果`JSON_EXTRACT()`。

  这是`->` MySQL 5.7中引入的列路径运算符的改进；`col->>"$.path"`相当于 `JSON_UNQUOTE(col->"$.path")`。内联路径运算符可以用来随时随地可以使用 `JSON_UNQUOTE(JSON_EXTRACT())`，如 `SELECT`列清单， `WHERE`和 `HAVING` 条款，并`ORDER BY`和 `GROUP BY`条款。有关更多信息，请参阅运算符的说明以及第12.16.8节“JSON路径语法”。

- 添加了两个JSON聚合函数 `JSON_ARRAYAGG()` 和 `JSON_OBJECTAGG()`。 `JSON_ARRAYAGG()` 将列或表达式作为其参数，并将结果聚合为单个`JSON`数组。表达式可以评估任何MySQL数据类型; 这不一定是一个`JSON`价值。 `JSON_OBJECTAGG()` 取两个列或表达式，它将其解释为键和值; 它将结果作为单个`JSON` 对象返回。有关更多信息和示例，请参见 第12.19节“聚合（GROUP BY）函数”。

- 添加了JSON实用程序功能 `JSON_PRETTY()`，`JSON` 以易于阅读的格式输出现有值; 每个JSON对象成员或数组值都打印在一个单独的行上，子对象或数组相对于其父对象是2个空格。

  此函数也适用于可以解析为JSON值的字符串。

  有关更多详细信息和示例，请参见 第12.16.7节“JSON实用程序函数”。

- `JSON`在使用查询 对值进行排序时`ORDER BY`，每个值现在由排序键的可变长度部分表示，而不是固定1K大小的一部分。在许多情况下，这可以减少过度使用; 例如，标量`INT`或偶 `BIGINT`数值实际上只需要很少的字节，因此该空间的剩余部分（最多90％或更多）由填充占用。此更改对性能具有以下好处：

  - 现在可以更有效地使用排序缓冲区空间，因此filesorts不需要尽早或者通常使用固定长度排序键刷新到磁盘。这意味着可以在内存中分类更多数据，从而避免不必要的磁盘访问。
  - 较短的密钥可以比较长的密钥更快地进行比较，从而显着提高性能。对于完全在内存中执行的排序以及需要写入和读取磁盘的排序，都是如此。

- 在MySQL 8.0.2中添加了对`JSON`列值的部分就地更新的支持，这比完全删除现有JSON值并在其位置编写新值更有效，就像之前更新任何`JSON`列时所做的那样 。要应用这种优化，更新，必须使用应用 `JSON_SET()`，`JSON_REPLACE()` 或 `JSON_REMOVE()`。无法将新元素添加到正在更新的JSON文档中; 文档中的值不会占用比更新前更多的空间。请参阅 JSON值的部分更新，详细讨论要求。

  可以将JSON文档的部分更新写入二进制日志，占用的空间比记录完整的JSON文档少。在使用基于语句的复制时，始终会记录部分更新。要使其与基于行的复制一起使用，必须先设置 `binlog_row_value_options=PARTIAL_JSON`; 有关更多信息，请参阅此变量的说明。

- 添加了JSON实用程序功能 `JSON_STORAGE_SIZE()` 和 `JSON_STORAGE_FREE()`。 `JSON_STORAGE_SIZE()` 在任何部分更新之前，返回用于JSON文档的二进制表示的字节存储空间（请参阅上一项）。 `JSON_STORAGE_FREE()` 显示`JSON`使用`JSON_SET()` 或 部分更新后的表格列中剩余的空间量 `JSON_REPLACE()`; 如果新值的二进制表示小于先前值的二进制表示，则大于零。

  这些函数中的每一个也接受JSON文档的有效字符串表示。对于此类值， `JSON_STORAGE_SIZE()` 在其转换为JSON文档后返回其二进制表示所使用的空间。对于包含JSON文档的字符串表示形式的变量， `JSON_STORAGE_FREE()` 返回零。如果无法将其（非null）参数解析为有效的JSON文档，并且`NULL`参数为，则 任一函数都会产生错误 `NULL`。

  有关更多信息和示例，请参见 第12.16.7节“JSON实用程序函数”。

  `JSON_STORAGE_SIZE()` 并 `JSON_STORAGE_FREE()` 在MySQL 8.0.2中实现。

- 在MySQL 8.0.2中添加了对`$[1 to 5]` XPath表达式等范围的支持 。此版本还为`last`关键字和相对寻址添加了支持，这样`$[last]`总是选择数组中的最后一个（编号最大的）元素和 `$[last-1]`最后一个元素。 `last`和使用它的表达

式也可以包含在范围定义中; 例如， `$[last-2 to last-1]`返回最后两个元素，但返回一个数组。有关其他信息和示例，请参阅 搜索和修改JSON值。

- 添加了旨在符合RFC 7396的JSON合并功能 。 `JSON_MERGE_PATCH()`，当在2个JSON对象上使用时，将它们合并为单个JSON对象，该对象具有以下集合的成员作为成员：
  - 第一个对象的每个成员，在第二个对象中没有成员具有相同的键。
  - 第二个对象的每个成员，其中没有成员在第一个对象中具有相同的键，并且其值不是JSON `null`文字。
  - 每个成员都有一个存在于两个对象中的键，并且其第二个对象中的值不是JSON `null`文字。

  作为此工作的一部分，该 `JSON_MERGE()`功能已重命名 `JSON_MERGE_PRESERVE()`。 `JSON_MERGE()`继续被认为是`JSON_MERGE_PRESERVE()`MySQL 8.0中的别名 ，但现在已被弃用，并且将在未来版本的MySQL中删除。

  有关更多信息和示例，请参见 第12.16.4节“修改JSON值的函数”。

- 实现了“ 最后重复密钥获胜 ” 重复密钥的规范化，与 RFC 7159和大多数JavaScript解析器一致。此处显示了此行为的示例，其中仅`x`保留具有密钥的最右侧成员：

```
mysql> SELECT JSON_OBJECT('x', '32', 'y', '[true, false]',
     >                    'x', '"abc"', 'x', '100') AS Result;
+-----------------------------------+
| Result                            |
+-----------------------------------+
| {"x": "100", "y": "[true, false]"} |
+-----------------------------------+
1 row in set (0.00 sec)
```

  插入MySQL `JSON`列的值 也以这种方式标准化，如下例所示：

```
mysql> CREATE TABLE t1 (c1 JSON);

mysql> INSERT INTO t1 VALUES ('{"x": 17, "x": "red", "x": [3, 5, 7]}');

mysql> SELECT c1 FROM t1;
+------------------+
| c1               |
+------------------+
| {"x": [3, 5, 7]} |
+------------------+
```

  这是与先前版本的MySQL不兼容的变化，其中 在这种情况下使用“ 第一次重复密钥获胜 ”算法。

  有关更多信息和示例，请参阅JSON值的规范化，合并和自动包装。

- `JSON_TABLE()` 在MySQL 8.0.4中 添加了该功能。此函数接受JSON数据并将其作为具有指定列的关系表返回。

  此函数具有语法 ，其中 是返回JSON数据的表达式，是应用于源的JSON路径， 是列定义的列表。这里显示一个例子： `JSON_TABLE(`***expr, path*** `COLUMNS` ***column_list***`) [AS]` ***alias***`)`***expr path column_list***

```
mysql> SELECT *
    -> FROM
    ->   JSON_TABLE(
```

```
    ->     '[{"a":3,"b":"0"},{"a":"3","b":"1"},{"a":2,"b":1},{"a":0},{"b":[1,2]}]'
    ->     "$[*]" COLUMNS(
    ->       rowid FOR ORDINALITY,
    ->
    ->       xa INT EXISTS PATH "$.a",
    ->       xb INT EXISTS PATH "$.b",
    ->
    ->       sa VARCHAR(100) PATH "$.a",
    ->       sb VARCHAR(100) PATH "$.b",
    ->
    ->       ja JSON PATH "$.a",
    ->       jb JSON PATH "$.b"
    ->     )
    ->   ) AS  jt1;
+-------+------+------+------+------+------+--------+
| rowid | xa   | xb   | sa   | sb   | ja   | jb     |
+-------+------+------+------+------+------+--------+
|     1 |    1 |    1 | 3    | 0    | 3    | "0"    |
|     2 |    1 |    1 | 3    | 1    | "3"  | "1"    |
|     3 |    1 |    1 | 2    | 1    | 2    | 1      |
|     4 |    1 |    0 | 0    | NULL | 0    | NULL   |
|     5 |    0 |    1 | NULL | NULL | NULL | [1, 2] |
+-------+------+------+------+------+------+--------+
```

JSON源表达式可以是生成有效JSON文档的任何表达式，包括JSON文字，表列或返回JSON的函数调用，如 `JSON_EXTRACT(t1, data, '$.post.comments')`。有关更多信息，请参见 第12.16.6节“JSON表函数”。

- **数据类型支持。** MySQL现在支持将表达式用作数据类型规范中的默认值。这包括使用表达式作为默认值 `BLOB`， `TEXT`， `GEOMETRY`，和 `JSON`数据类型，这在以前是根本不会被分配缺省值。有关详细信息，请参见第11.7节“数据类型默认值”。

- **优化。** 添加了以下优化程序增强功能：

  - MySQL现在支持隐形索引。优化器根本不使用不可见索引，但以其他方式正常维护。默认情况下，索引是可见的。不可见索引可以测试删除索引对查询性能的影响，而不会进行破坏性更改，如果索引结果是必需的，则必须撤消。请参见 第8.3.12节“不可见索引”。

  - MySQL现在支持降序索引： `DESC`在索引定义中不再被忽略，但会导致按键降序存储键值。以前，索引可以按相反顺序扫描，但性能会受到影响。可以按正向顺序扫描降序索引，这样更有效。当最有效的扫描顺序混合某些列的升序和其他列的降序时，降序索引还使优化器可以使用多列索引。请参见第8.3.13节“降序索引”。

  - MySQL现在支持创建索引表达式值而不是列值的功能索引键部分。功能键部件支持索引无法索引的值，例如 `JSON` 值。有关详细信息，请参见第13.1.14节“CREATE INDEX语法”。

- **公用表表达式。** MySQL现在支持非递归和递归的公用表表达式。公用表表达式允许使用命名的临时结果集，通过允许 `WITH`语句之前的子句`SELECT`和某些其他语句来实现。有关更多信息，请参见 第13.2.13节“WITH语法（公用表表达式）”。

- **窗口功能。** MySQL现在支持窗口函数，对于查询中的每一行，它使用与该行相关的行执行计算。这些包括诸如 `RANK()`， `LAG()`，和 `NTILE()`。此外，现在可以将几个现有的聚合函数用作窗口函数; 例如， `SUM()`和 `AVG()`。有关更多信息，请参见第12.20节“窗口函数”。

- **正则表达式支持。** 以前，MySQL使用Henry Spencer正则表达式库来支持正则表达式运算符（ `REGEXP`， `RLIKE`）。正则表达式支持已使用国际Unicode组件（ICU）重新实现，它提供完整的Unicode支持并且是多字节安全的。该 `REGEXP_LIKE()`函数以`REGEXP`和 `RLIKE` 运算符的方式执行正则表达式匹配，这些 运算符现在是该函数的同义词。此外， `REGEXP_INSTR()`， `REGEXP_REPLACE()`，和 `REGEXP_SUBSTR()`函数可用于查找匹配位置并分别执行子串替换和

提取。该 regexp_stack_limit和 regexp_time_limit系统变量提供了通过发动机匹配的资源消耗的控制。有关更多信息，请参见 第12.5.2节“正则表达式”。有关使用正则表达式的应用程序可能受实现更改影响的方式的信息，请参阅 正则表达式兼容性注意事项。

- **内部临时表。** 的TempTable存储引擎替换MEMORY存储引擎作为默认发动机用于在内存中的内部临时表。该TempTable存储引擎提供了有效的存储 VARCHAR和 VARBINARY列。的 internal_tmp_mem_storage_engine 会话变量定义了用于在存储器内的临时表的存储引擎。允许的值是 TempTable（默认值）和 MEMORY。的 temptable_max_ram 配置选项定义的存储器，所述最大数量TempTable之前的数据被存储到磁盘存储引擎可以使用。

- **日志记录。** 重写错误日志记录以使用MySQL组件体系结构。传统错误日志记录使用内置组件实现，使用系统日志进行日志记录实现为可加载组件。此外，还提供可加载的JSON日志写入程序。要控制要启用的日志组件，请使用 log_error_services系统变量。有关更多信息，请参见 第5.4.2节“错误日志”。

- **备份锁。** 新类型的备份锁在联机备份期间允许DML，同时防止可能导致快照不一致的操作。新的备份锁由LOCK INSTANCE FOR BACKUP 和 UNLOCK INSTANCE语法支持 。该 BACKUP_ADMIN权限才能使用这些语句。

- **复制。** 对MySQL Replication进行了以下增强：

  - MySQL Replication现在支持使用紧凑的二进制格式对JSON文档的部分更新进行二进制日志记录，从而节省了日志记录完整JSON文档的空间。当使用基于语句的日志记录时，这种紧凑日志记录会自动完成，并且可以通过将新 binlog_row_value_options系统变量设置为来启用 PARTIAL_JSON。有关更多信息，请参阅JSON值的部分更新以及 binlog_row_value_options。

功能在MySQL 8.0中不推荐使用

MySQL 8.0中不推荐使用以下功能，可能会在将来的系列中删除。在显示替代方案的地方，应更新应用程序以使用它们。

对于使用MySQL 8.0中已弃用的已在较高MySQL系列中删除的功能的应用程序，从MySQL 8.0主服务器复制到更高级别的从服务器时语句可能会失败，或者可能对主服务器和从服务器产生不同的影响。为避免此类问题，应修改使用8.0中不推荐使用的功能的应用程序以避免它们，并尽可能使用备选方案。

- utf8mb3不推荐使用 该字符集。请utf8mb4改用。

- 该validate_password插件已重新实现以使用服务器组件基础结构。插件形式validate_password仍然可用，但已弃用，将在未来的MySQL版本中删除。使用该插件的MySQL安装应该转换为使用该组件。请参见 第6.5.3.3节“转换到密码验证组件”。

- 该ALTER TABLESPACE和 条款已被弃用。 DROP TABLESPACE ENGINE

- 在 PAD_CHAR_TO_FULL_LENGTH SQL模式已经过时了。

- 该JSON_MERGE()函数已弃用。请 JSON_MERGE_PRESERVE() 改用。

- 自MySQL 8.0.13起，不推荐使用对 支持TABLESPACE = innodb_file_per_table和TABLESPACE = innodb_temporary子句的 支持CREATE TEMPORARY TABLE。

MySQL 8.0中删除的功能

以下项目已过时，已在MySQL 8.0中删除。在显示替代方案的地方，应更新应用程序以使用它们。

对于使用MySQL 8.0中删除的功能的MySQL 5.7应用程序，从MySQL 5.7主服务器复制到MySQL 8.0从服务器时语句可能会失败，或者可能对主服务器和从服务器产生不同的影响。为避免此类问题，应修改使用MySQL 8.0中删除的功能的应用程序以避免它们并尽可能使用替代方案。

在innodb_locks_unsafe_for_binlog除去系统变量。该READ COMMITTED隔离级别提供了类似的功能。

- information_schema_stats MySQL 8.0.0中引入 的配置选项已被删除，并information_schema_stats_expiry 在MySQL 8.0.3中替换 。

  information_schema_stats_expiry定义缓存INFORMATION_SCHEMA表统计信息的到期设置 。有关更多信息，请参见第8.2.3节“优化INFORMATION_SCHEMA查询”。

- InnoDB在MySQL 8.0.3中删除了 与已废弃系统表相关的代码。 INFORMATION_SCHEMA基于InnoDB系统表的视图被数据字典表上的内部系统视图替换。受影响的 InnoDB INFORMATION_SCHEMA视图已重命名：

  表1.1重命名的InnoDB信息模式视图

旧名	新名字
INNODB_SYS_COLUMNS	INNODB_COLUMNS
INNODB_SYS_DATAFILES	INNODB_DATAFILES
INNODB_SYS_FIELDS	INNODB_FIELDS
INNODB_SYS_FOREIGN	INNODB_FOREIGN
INNODB_SYS_FOREIGN_COLS	INNODB_FOREIGN_COLS
INNODB_SYS_INDEXES	INNODB_INDEXES
INNODB_SYS_TABLES	INNODB_TABLES
INNODB_SYS_TABLESPACES	INNODB_TABLESPACES
INNODB_SYS_TABLESTATS	INNODB_TABLESTATS
INNODB_SYS_VIRTUAL	INNODB_VIRTUAL

  升级到MySQL 8.0.3或更高版本后，更新引用先前InnoDB INFORMATION_SCHEMA视图名称的所有脚本。

- 已删除与帐户管理相关的以下功能：

  - 使用GRANT创建用户。相反，使用CREATE USER。遵循这种做法使得 NO_AUTO_CREATE_USERSQL模式对于GRANT 语句来说无关紧要，因此它也被删除了。
  - 使用GRANT修改不是权限指派其他帐户属性。这包括身份验证，SSL和资源限制属性。相反，在帐户创建时建立这样的属性，CREATE USER然后用它们修改它们 ALTER USER。
  - IDENTIFIED BY PASSWORD '***hash_string***'CREATE USER 和的 语法GRANT。相反，使用 for 和 ，其中 值是与指定插件兼容的格式。 IDENTIFIED WITH ***auth_plugin*** AS '***hash_string***'CREATE USERALTER USER'***hash_string***'

    此外，由于IDENTIFIED BY PASSWORD语法已被删除，因此 log_builtin_as_identified_by_password 系统变量是多余的，并已被删除。

  - 该PASSWORD()功能。此外，PASSWORD()删除意味着 语法不再可用。 SET PASSWORD ... = PASSWORD('***auth_string***')
  - 该old_passwords 系统变量。

- 查询缓存已被删除。删除包括以下项目：

  - 该FLUSH QUERY CACHE和 RESET QUERY CACHE语句。
  - 这些系统变量： query_cache_limit ， query_cache_min_res_unit ， query_cache_size ， query_cache_type ， query_cache_wlock_invalidate。

- 这些状态变量： Qcache_free_blocks , Qcache_free_memory , Qcache_hits , Qcache_inserts , Qcache_lowmem_prunes , Qcache_not_cached , Qcache_queries_in_cache , Qcache_total_blocks。
- 这些线程状态：checking privileges on cached query , checking query cache for query , invalidating query cache entries , sending cached result to client , storing result in query cache , Waiting for query cache lock。
- 该SQL_CACHE SELECT修改。

这些已弃用的查询缓存项目仍然不推荐使用，但不起作用，将在以后的MySQL版本中删除：

- SQL_NO_CACHE SELECT 修改。
- 该ndb_cache_check_time系统变量。

该have_query_cache系统变量仍然是过时的，总是有一个值 NO，并会在将来的MySQL版本中删除。

- 数据字典提供有关数据库对象的信息，因此服务器不再检查数据目录中的目录名以查找数据库。因此， --ignore-db-dir选项和 ignore_db_dirs系统变量是无关的并且已被删除。
- 该tx_isolation和 tx_read_only系统变量已被删除。使用 transaction_isolation和 transaction_read_only 替代。
- 该sync_frm系统变量已被删除，因为.frm文件已经过时。
- 该secure_auth系统变量和 --secure-auth客户端选项已被删除。已删除C API函数的MYSQL_SECURE_AUTH选项 mysql_options()。
- 该multi_range_count系统变量已被删除。
- 该log_warnings系统变量和 --log-warnings服务器选项已被删除。请改用 log_error_verbosity系统变量。
- sql_log_bin已删除系统变量 的全局范围 。sql_log_bin只有会话范围，@@global.sql_log_bin应该调整依赖访问的应用程序 。
- 该metadata_locks_cache_size和 metadata_locks_hash_instances系统变量已被删除。
- 未使用date_format , datetime_format , time_format , 和 max_tmp_tables系统变量已被删除。
- 这些过时兼容性SQL模式已被删除： DB2 , MAXDB , MSSQL , MYSQL323 , MYSQL40 , ORACLE , POSTGRESQL , NO_FIELD_OPTIONS , NO_KEY_OPTIONS , NO_TABLE_OPTIONS。它们不能再分配给sql_mode系统变量或用作 **mysqldump** --compatible选项的允许值 。

  删除MAXDB意味着 TIMESTAMP数据类型为 CREATE TABLE或被 ALTER TABLE视为 TIMESTAMP，并且不再被视为 DATETIME。
- 已删除子句的已 弃用ASC或 DESC限定符GROUP BY。以前依赖于GROUP BY排序的查询可能会产生与以前的MySQL版本不同的结果。要生成给定的排序顺序，请提供一个ORDER BY子句。
- 该语句的EXTENDED和 PARTITIONS关键字 EXPLAIN已被删除。这些关键字是不必要的，因为它们的效果总是启用
- 这些与加密相关的项目已被删除：
  - 该ENCODE()和 DECODE()功能。

  - 该ENCRYPT()功能。
  - 的DES_ENCRYPT()，和 DES_DECRYPT()功能的--des-key-file 选项，have_crypt 系统变量，则DES_KEY_FILE该选项FLUSH 语句和HAVE_CRYPT **CMake的**选项。

  取代已删除的加密函数：对于 ENCRYPT()，请考虑使用 SHA2()单向散列。对于其他人，请考虑使用 AES_ENCRYPT()而 AES_DECRYPT()不是。

- 在MySQL 5.7中，不推荐使用多个名称下的多个空间函数，以便使空间函数名称空间更加一致，目标是每个空间函数名称ST_在执行精确操作时开始，或者MBR如果它执行了基于最小边界矩形的操作。在MySQL 8.0中，废弃的函数被去除仅离开对应的ST_和 MBR功能：
  - 这些功能有利于去除的 MBR名字： Contains() , Disjoint() , Equals() , Intersects() , Overlaps() , Within()。
  - 这些功能有利于去除的 ST_名字：Area() , AsBinary() , AsText() , AsWKB() , AsWKT() , Buffer() , Centroid() , ConvexHull() , Crosses() , Dimension() , Distance() , EndPoint() , Envelope() , ExteriorRing() , GeomCollFromText() , GeomCollFromWKB() , GeomFromText() , GeomFromWKB() , GeometryCollectionFromText() , GeometryCollectionFromWKB() , GeometryFromText() , GeometryFromWKB() , GeometryN() , GeometryType() , InteriorRingN() , IsClosed() , IsEmpty() , IsSimple() , LineFromText() , LineFromWKB() , LineStringFromText() , LineStringFromWKB() , MLineFromText() , MLineFromWKB() , MPointFromText() , MPointFromWKB() , MPolyFromText() , MPolyFromWKB() , MultiLineStringFromText() , MultiLineStringFromWKB() , MultiPointFromText() , MultiPointFromWKB() , MultiPolygonFromText() , MultiPolygonFromWKB() , NumGeometries() , NumInteriorRings() , NumPoints() , PointFromText() , PointFromWKB() , PointN() , PolyFromText() , PolyFromWKB() , PolygonFromText() , PolygonFromWKB() , SRID() , StartPoint() , Touches() , X() , Y()。
  - GLength()被删除有利于 ST_Length()。
- 第12.15.4节“从WKB值创建几何值 的函数”中描述的函数 以前接受WKB字符串或几何参数。不再允许使用几何参数并产生错误。有关使用几何参数迁移查询的指南，请参阅该部分。
- 解析器不再被视为SQL语句中\N的同义词NULL。请 NULL改用。

  这种变化并不影响执行文本文件导入和导出操作 LOAD DATA INFILE或 SELECT ... INTO OUTFILE用于其NULL 继续受到代表\N。请参见 第13.2.7节“LOAD DATA INFILE语法”。
- PROCEDURE ANALYSE() 语法被删除。
- 客户端--ssl和 --ssl-verify-server-cert选项已被删除。使用 --ssl-mode=REQUIRED而不是--ssl=1或 --enable-ssl。使用 --ssl-mode=DISABLED替代--ssl=0，--skip-ssl或 --disable-ssl。使用 --ssl-mode=VERIFY_IDENTITY 而不是--ssl-verify-server-cert 选项。（服务器端 --ssl选项保持不变。）

  对于C API，MYSQL_OPT_SSL_ENFORCE以及 MYSQL_OPT_SSL_VERIFY_SERVER_CERT选项mysql_options() 对应于客户端--ssl和 --ssl-verify-server-cert选择和已被删除。使用MYSQL_OPT_SSL_MODE选项值SSL_MODE_REQUIRED 或 SSL_MODE_VERIFY_IDENTITY代替。
- 该--temp-pool服务器选项已被删除。
- 该--ignore-builtin-innodb 服务器选项和 ignore_builtin_innodb 系统变量已被删除。

- 服务器不再通过添加#mysql50#前缀将包含特殊字符的MySQL 5.1之前的数据库名称转换为5.1格式。由于不再执行这些转换，因此 已删除了**mysqlcheck**和语句的子句 以及状态变量的选项--fix-db-names和 --fix-table-names选项 。 UPGRADE DATA DIRECTORY NAMEALTER DATABASECom_alter_db_upgrade

  仅从一个主要版本到另一个主要版本（例如，5.0到5.1或5.1到5.5）支持升级，因此将旧的5.0数据库名称转换为当前版本的MySQL几乎不需要。作为解决方法，在升级到更新版本之前，将MySQL 5.0安装升级到MySQL 5.1。

- 该**mysql_install_db的**计划已经从MySQL分发中删除。应该通过使用 或 选项调用**mysqld**来执行数据目录初始化 。此外，已删除**mysql_install_db**使用的 **mysqld** 选项，并且已删除控制**mysql_install_db**的安装位置的 **选项**。 --initialize--initialize-insecure--bootstrapINSTALL_SCRIPTDIR CMake

- 通用分区处理程序已从MySQL服务器中删除。为了支持给定表的分区，用于表的存储引擎现在必须提供其自己的（“本机”）分区处理程序。这些--partition和 --skip-partition选项已从MySQL服务器中删除，与分区相关的条目不再显示在表的输出中SHOW PLUGINS或 INFORMATION_SCHEMA.PLUGINS 表中。

  两个MySQL存储引擎目前提供本机分区支持 - InnoDB和 NDB; 其中，仅 InnoDB在MySQL 8.0中受支持。任何使用任何其他存储引擎在MySQL 8.0中创建分区表的尝试都会失败。

  升级的分歧。 使用比其他的存储引擎分区表的直接升级InnoDB（如 MyISAM不支持）从MySQL 5.7（或更早）到MySQL 8.0。处理此类表有两种选择：

  - 使用，删除表的分区 ALTER TABLE ... REMOVE PARTITIONING。
  - 更改用于表的存储引擎 InnoDB，用 ALTER TABLE ... ENGINE=INNODB。

  在InnoDB 将服务器升级到MySQL 8.0之前，必须为每个分区的非表执行刚刚列出的两个操作中的至少一个。否则，升级后无法使用此类表。

  由于表创建语句导致使用没有分区支持的存储引擎的分区表现在失败并出现错误（ER_CHECK_NOT_IMPLEMENTED），因此必须确保转储文件中的任何语句（例如**mysqldump**编写的语句）从您希望导入到创建分区表的MySQL 8.0服务器的旧版MySQL中，也不会指定存储引擎，例如MyISAM没有本机分区处理程序的存储引擎 。您可以通过执行以下任一操作来执行此操作：

  - 从除CREATE TABLE使用除STORAGE ENGINE选项之外的值的语句中删除对分区的任何引用 InnoDB。
  - 默认情况下InnoDB， 将存储引擎指定为 或允许 InnoDB用作表的存储引擎。

  有关更多信息，请参见 第22.6.2节“分区与存储引擎相关的限制”。

- 系统和状态变量信息不再保留在INFORMATION_SCHEMA。这些表已被删除： GLOBAL_VARIABLES， SESSION_VARIABLES， GLOBAL_STATUS， SESSION_STATUS。请改用相应的性能架构表。请参见 第25.11.13节“性能模式系统变量表”和 第25.11.14节“性能模式状态变量表”。此外，show_compatibility_56 系统变量已被删除。它用于过渡期间系统和状态变量信息INFORMATION_SCHEMA表已移至Performance Schema表，不再需要。这些状态变量已被删除： Slave_heartbeat_period， Slave_last_heartbeat， Slave_received_heartbeats， Slave_retried_transactions， Slave_running。他们提供的信息可在性能模式表中找到; 请参阅 迁移到性能架构系统和状态变量表。

- 性能模式setup_timers表已被删除，表中的TICK行也已删除performance_timers。

- 该libmysqld嵌入式服务器库已被删除，连同：

  - 在， ， ，和 选项 mysql_options() MYSQL_OPT_GUESS_CONNECTIONMYSQL_OPT_USE_EMBEDDED_CONNECTIONMYSQL_OPT_USE_REMOTE_CONNECTION

`MYSQL_SET_CLIENT_IP`

  - 该**mysql_config** `--libmysqld-libs`，`--embedded-libs`和 `--embedded`选项
  - 该**CMake的** `WITH_EMBEDDED_SERVER`，`WITH_EMBEDDED_SHARED_LIBRARY`和 `INSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR` 选项
  - （未记录的）**mysql** `--server-arg`选项
  - 该**mysqltest** `--embedded-server`，`--server-arg`和 `--server-file`选项
  - 该**mysqltest_embedded**和 **mysql_client_test_embedded**测试程序

- 该**mysql_plugin**工具已被删除。替代方案包括使用`--plugin-load`or `--plugin-load-add`选项在服务器启动时加载插件，或者在运行时使用`INSTALL PLUGIN`语句加载插件 。
- 以下服务器错误代码未使用且已被删除。应更新专门针对任何这些错误进行测试的应用程序。

```
ER_BINLOG_READ_EVENT_CHECKSUM_FAILURE
ER_BINLOG_ROW_RBR_TO_SBR
ER_BINLOG_ROW_WRONG_TABLE_DEF
ER_CANT_ACTIVATE_LOG
ER_CANT_CHANGE_GTID_NEXT_IN_TRANSACTION
ER_CANT_CREATE_FEDERATED_TABLE
ER_CANT_CREATE_SROUTINE
ER_CANT_DELETE_FILE
ER_CANT_GET_WD
ER_CANT_SET_GTID_PURGED_WHEN_GTID_MODE_IS_OFF
ER_CANT_SET_WD
ER_CANT_WRITE_LOCK_LOG_TABLE
ER_CREATE_DB_WITH_READ_LOCK
ER_CYCLIC_REFERENCE
ER_DB_DROP_DELETE
ER_DELAYED_NOT_SUPPORTED
ER_DIFF_GROUPS_PROC
ER_DISK_FULL
ER_DROP_DB_WITH_READ_LOCK
ER_DROP_USER
ER_DUMP_NOT_IMPLEMENTED
ER_ERROR_DURING_CHECKPOINT
ER_ERROR_ON_CLOSE
ER_EVENTS_DB_ERROR
ER_EVENT_CANNOT_DELETE
ER_EVENT_CANT_ALTER
ER_EVENT_COMPILE_ERROR
ER_EVENT_DATA_TOO_LONG
ER_EVENT_DROP_FAILED
ER_EVENT_MODIFY_QUEUE_ERROR
ER_EVENT_NEITHER_M_EXPR_NOR_M_AT
ER_EVENT_OPEN_TABLE_FAILED
ER_EVENT_STORE_FAILED
ER_EXEC_STMT_WITH_OPEN_CURSOR
ER_FAILED_ROUTINE_BREAK_BINLOG
ER_FLUSH_MASTER_BINLOG_CLOSED
ER_FORM_NOT_FOUND
ER_FOUND_GTID_EVENT_WHEN_GTID_MODE_IS_OFF__UNUSED
ER_FRM_UNKNOWN_TYPE
ER_GOT_SIGNAL
ER_GRANT_PLUGIN_USER_EXISTS
```

```
ER_GTID_MODE_REQUIRES_BINLOG
ER_GTID_NEXT_IS_NOT_IN_GTID_NEXT_LIST
ER_HASHCHK
ER_INDEX_REBUILD
ER_INNODB_NO_FT_USES_PARSER
ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR
ER_LOAD_DATA_INVALID_COLUMN_UNUSED
ER_LOGGING_PROHIBIT_CHANGING_OF
ER_MALFORMED_DEFINER
ER_MASTER_KEY_ROTATION_ERROR_BY_SE
ER_NDB_CANT_SWITCH_BINLOG_FORMAT
ER_NEVER_USED
ER_NISAMCHK
ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR
ER_NO_FILE_MAPPING
ER_NO_GROUP_FOR_PROC
ER_NO_RAID_COMPILED
ER_NO_SUCH_KEY_VALUE
ER_NO_SUCH_PARTITION__UNUSED
ER_OBSOLETE_CANNOT_LOAD_FROM_TABLE
ER_OBSOLETE_COL_COUNT_DOESNT_MATCH_CORRUPTED
ER_ORDER_WITH_PROC
ER_PARTITION_SUBPARTITION_ERROR
ER_PARTITION_SUBPART_MIX_ERROR
ER_PART_STATE_ERROR
ER_PASSWD_LENGTH
ER_QUERY_ON_MASTER
ER_RBR_NOT_AVAILABLE
ER_SKIPPING_LOGGED_TRANSACTION
ER_SLAVE_CHANNEL_DELETE
ER_SLAVE_MULTIPLE_CHANNELS_HOST_PORT
ER_SLAVE_MUST_STOP
ER_SLAVE_WAS_NOT_RUNNING
ER_SLAVE_WAS_RUNNING
ER_SP_GOTO_IN_HNDLR
ER_SP_PROC_TABLE_CORRUPT
ER_SQL_MODE_NO_EFFECT
ER_SR_INVALID_CREATION_CTX
ER_TABLE_NEEDS_UPG_PART
ER_TOO_MUCH_AUTO_TIMESTAMP_COLS
ER_UNEXPECTED_EOF
ER_UNION_TABLES_IN_DIFFERENT_DIR
ER_UNSUPPORTED_BY_REPLICATION_THREAD
ER_UNUSED1
ER_UNUSED2
ER_UNUSED3
ER_UNUSED4
ER_UNUSED5
ER_UNUSED6
ER_VIEW_SELECT_DERIVED_UNUSED
ER_WRONG_MAGIC
ER_WSAS_FAILED
```

- 已弃用的表`INFORMATION_SCHEMA` `INNODB_LOCKS`和 `INNODB_LOCK_WAITS`表已被删除。请改用性能模式 `data_locks`和 `data_lock_waits`表。

> **注意**

> 在MySQL 5.7中，LOCK_TABLE在列INNODB_LOCKS表和locked_table在列 sys模式 innodb_lock_waits和 x$innodb_lock_waits视图包含组合模式/表名的值。在MySQL 8.0中，data_locks表和 sys模式视图包含单独的模式名称和表名称列。请参见 第26.4.3.9 节“innodb_lock_waits和x $ innodb_lock_waits视图”。

- InnoDB不再支持压缩的临时表。当 innodb_strict_mode启用（默认值），CREATE TEMPORARY TABLE如果返回错误 ROW_FORMAT=COMPRESSED或 KEY_BLOCK_SIZE指定的。如果 innodb_strict_mode禁用，则发出警告，并使用非压缩行格式创建临时表。
- InnoDB 在MySQL数据目录之外创建表空间数据文件时，不再创建 .isl文件（InnoDB符号链接文件）。该 innodb_directories选项现在支持查找在数据目录之外创建的表空间文件。

  通过此更改，.isl不再支持在服务器脱机时通过手动修改文件来移动远程表空间 。此innodb_directories选项现在支持移动远程表空间文件 。请参见第15.7.7节“在服务器脱机时移动表空间文件”。
- InnoDB删除了 以下文件格式配置选项：
  - innodb_file_format
  - innodb_file_format_check
  - innodb_file_format_max
  - innodb_large_prefix

  文件格式配置选项对于创建与InnoDBMySQL 5.1 早期版本兼容的表是必需的 。现在MySQL 5.1已经到了产品生命周期的末尾，不再需要这些选项。

  该FILE_FORMAT列已从INNODB_TABLES和 INNODB_TABLESPACES信息模式表中删除。
- 的innodb_support_xa系统变量，这使得能够对两相支持XA事务的提交，除去。InnoDB始终启用对XA事务中的两阶段提交的支持。
- 已删除对DTrace的支持。
- 该JSON_APPEND()功能已被删除。请 JSON_ARRAY_APPEND()改用。
- InnoDBMySQL 8.0.13中删除了 在共享表空间中放置表分区的支持 。共享表空间包括 InnoDB系统表空间和通用表空间。有关识别共享表空间中的分区并将其移动到每个表文件表空间的信息，请参见第2.11.1.4节“准备升级安装”。
- 支持SET 在MySQL 8.0.13中不推荐使用的语句中设置用户变量 。此功能可能会在MySQL 9.0中删除。

1.5 MySQL 8.0中添加，弃用或删除的服务器和状态变量和选项

本节列出了第一次添加，已弃用或已在MySQL 8.0中删除的服务器变量，状态变量和选项。

- 选项和变量在MySQL 8.0中引入
- 选项和变量在MySQL 8.0中不推荐使用
- MySQL 8.0中删除的选项和变量

选项和变量在MySQL 8.0中引入

以下系统变量，状态变量和选项是MySQL 8.0中的新增功能，并未包含在任何先前的发行版系列中。

- `Acl_cache_items_count`：在MySQL 8.0.0中添加。
- `Audit_log_current_size`：在MySQL 8.0.11中添加。
- `Audit_log_event_max_drop_size`：在MySQL 8.0.11中添加。
- `Audit_log_events`：在MySQL 8.0.11中添加。
- `Audit_log_events_filtered`：在MySQL 8.0.11中添加。
- `Audit_log_events_lost`：在MySQL 8.0.11中添加。
- `Audit_log_events_written`：在MySQL 8.0.11中添加。
- `Audit_log_total_size`：在MySQL 8.0.11中添加。
- `Audit_log_write_waits`：在MySQL 8.0.11中添加。
- `Caching_sha2_password_rsa_public_key`：在MySQL 8.0.4中添加。
- `Com_alter_resource_group`：在MySQL 8.0.3中添加。
- `Com_alter_user_default_role`：在MySQL 8.0.0中添加。
- `Com_create_resource_group`：在MySQL 8.0.3中添加。
- `Com_create_role`：在MySQL 8.0.0中添加。
- `Com_drop_resource_group`：在MySQL 8.0.3中添加。
- `Com_drop_role`：在MySQL 8.0.0中添加。

- `Com_grant_roles`：在MySQL 8.0.0中添加。
- `Com_install_component`：在MySQL 8.0.0中添加。
- `Com_revoke_roles`：在MySQL 8.0.0中添加。
- `Com_set_resource_group`：在MySQL 8.0.3中添加。
- `Com_set_role`：在MySQL 8.0.0中添加。
- `Com_uninstall_component`：在MySQL 8.0.0中添加。
- `Connection_control_delay_generated`：在MySQL 8.0.1中添加。
- `Firewall_access_denied`：在MySQL 8.0.11中添加。
- `Firewall_access_granted`：在MySQL 8.0.11中添加。
- `Firewall_cached_entries`：在MySQL 8.0.11中添加。
- `Secondary_engine_execution_count`：在MySQL 8.0.13中添加。
- `activate_all_roles_on_login`：在MySQL 8.0.2中添加。
- `audit-log`：在MySQL 8.0.11中添加。
- `audit_log_buffer_size`：在MySQL 8.0.11中添加。
- `audit_log_compression`：在MySQL 8.0.11中添加。
- `audit_log_connection_policy`：在MySQL 8.0.11中添加。
- `audit_log_current_session`：在MySQL 8.0.11中添加。
- `audit_log_encryption`：在MySQL 8.0.11中添加。
- `audit_log_exclude_accounts`：在MySQL 8.0.11中添加。
- `audit_log_file`：在MySQL 8.0.11中添加。
- `audit_log_filter_id`：在MySQL 8.0.11中添加。
- `audit_log_flush`：在MySQL 8.0.11中添加。
- `audit_log_format`：在MySQL 8.0.11中添加。
- `audit_log_include_accounts`：在MySQL 8.0.11中添加。
- `audit_log_policy`：在MySQL 8.0.11中添加。

- `audit_log_read_buffer_size`：在MySQL 8.0.11中添加。
- `audit_log_rotate_on_size`：在MySQL 8.0.11中添加。
- `audit_log_statement_policy`：在MySQL 8.0.11中添加。
- `audit_log_strategy`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_auth_method_name`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_bind_base_dn`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_bind_root_dn`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_bind_root_pwd`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_ca_path`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_group_search_attr`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_group_search_filter`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_init_pool_size`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_log_status`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_max_pool_size`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_server_host`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_server_port`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_tls`：在MySQL 8.0.11中添加。
- `authentication_ldap_sasl_user_search_attr`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_auth_method_name`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_bind_base_dn`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_bind_root_dn`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_bind_root_pwd`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_ca_path`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_group_search_attr`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_group_search_filter`：在MySQL 8.0.11中添加。

- `authentication_ldap_simple_init_pool_size`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_log_status`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_max_pool_size`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_server_host`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_server_port`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_tls`：在MySQL 8.0.11中添加。
- `authentication_ldap_simple_user_search_attr`：在MySQL 8.0.11中添加。
- `authentication_windows_log_level`：在MySQL 8.0.11中添加。
- `authentication_windows_use_principal_name`：在MySQL 8.0.11中添加。
- `binlog_expire_logs_seconds`：在MySQL 8.0.1中添加。
- `binlog_row_metadata`：在MySQL 8.0.1中添加。
- `binlog_row_value_options`：在MySQL 8.0.3中添加。
- `binlog_transaction_dependency_history_size`：在MySQL 8.0.1中添加。
- `binlog_transaction_dependency_tracking`：在MySQL 8.0.1中添加。
- `caching_sha2_password_auto_generate_rsa_keys`：在MySQL 8.0.4中添加。
- `caching_sha2_password_private_key_path`：在MySQL 8.0.3中添加。
- `caching_sha2_password_public_key_path`：在MySQL 8.0.3中添加。
- `connection_control_failed_connections_threshold`：在MySQL 8.0.1中添加。
- `connection_control_max_connection_delay`：在MySQL 8.0.1中添加。
- `connection_control_min_connection_delay`：在MySQL 8.0.1中添加。
- `cte_max_recursion_depth`：在MySQL 8.0.3中添加。
- `default_collation_for_utf8mb4`：在MySQL 8.0.11中添加。
- `dragnet.Status`：在MySQL 8.0.12中添加。
- `dragnet.log_error_filter_rules`：在MySQL 8.0.4中添加。
- `early-plugin-load`：在MySQL 8.0.0中添加。

- group_replication_communication_debug_options：在MySQL 8.0.3中添加。
- group_replication_flow_control_hold_percent：在MySQL 8.0.2中添加。
- group_replication_flow_control_max_commit_quota：在MySQL 8.0.2中添加。
- group_replication_flow_control_member_quota_percent：在MySQL 8.0.2中添加。
- group_replication_flow_control_min_quota：在MySQL 8.0.2中添加。
- group_replication_flow_control_min_recovery_quota：在MySQL 8.0.2中添加。
- group_replication_flow_control_period：在MySQL 8.0.2中添加。
- group_replication_flow_control_release_percent：在MySQL 8.0.2中添加。
- group_replication_member_expel_timeout：在MySQL 8.0.13中添加。
- group_replication_member_weight：在MySQL 8.0.2中添加。
- group_replication_recovery_get_public_key：在MySQL 8.0.4中添加。
- group_replication_recovery_public_key_path：在MySQL 8.0.4中添加。
- group_replication_unreachable_majority_timeout：在MySQL 8.0.2中添加。
- histogram_generation_max_mem_size：在MySQL 8.0.2中添加。
- information_schema_stats_expiry：在MySQL 8.0.3中添加。
- innodb_buffer_pool_debug：在MySQL 8.0.0中添加。
- innodb_buffer_pool_in_core_file：在MySQL 8.0.14中添加。
- innodb_checkpoint_disabled：在MySQL 8.0.2中添加。
- innodb_ddl_log_crash_reset_debug：在MySQL 8.0.3中添加。
- innodb_deadlock_detect：在MySQL 8.0.0中添加。
- innodb_dedicated_server：在MySQL 8.0.3中添加。
- innodb_directories：在MySQL 8.0.4中添加。
- innodb_fsync_threshold：在MySQL 8.0.13中添加。
- innodb_log_checkpoint_fuzzy_now：在MySQL 8.0.13中添加。
- innodb_log_spin_cpu_abs_lwm：在MySQL 8.0.11中添加。

- `innodb_log_spin_cpu_pct_hwm`：在MySQL 8.0.11中添加。
- `innodb_log_wait_for_flush_spin_hwm`：在MySQL 8.0.11中添加。
- `innodb_parallel_read_threads`：在MySQL 8.0.14中添加。
- `innodb_print_ddl_logs`：在MySQL 8.0.3中添加。
- `innodb_redo_log_encrypt`：在MySQL 8.0.1中添加。
- `innodb_scan_directories`：在MySQL 8.0.2中添加。
- `innodb_stats_include_delete_marked`：在MySQL 8.0.1中添加。
- `innodb_temp_tablespaces_dir`：在MySQL 8.0.13中添加。
- `innodb_tmpdir`：在MySQL 8.0.0中添加。
- `innodb_undo_log_encrypt`：在MySQL 8.0.1中添加。
- `internal_tmp_mem_storage_engine`：在MySQL 8.0.2中添加。
- `keyring-migration-destination`：在MySQL 8.0.4中添加。
- `keyring-migration-host`：在MySQL 8.0.4中添加。
- `keyring-migration-password`：在MySQL 8.0.4中添加。
- `keyring-migration-port`：在MySQL 8.0.4中添加。
- `keyring-migration-socket`：在MySQL 8.0.4中添加。
- `keyring-migration-source`：在MySQL 8.0.4中添加。
- `keyring-migration-user`：在MySQL 8.0.4中添加。
- `keyring_aws_cmk_id`：在MySQL 8.0.11中添加。
- `keyring_aws_conf_file`：在MySQL 8.0.11中添加。
- `keyring_aws_data_file`：在MySQL 8.0.11中添加。
- `keyring_aws_region`：在MySQL 8.0.11中添加。
- `keyring_encrypted_file_data`：在MySQL 8.0.11中添加。
- `keyring_encrypted_file_password`：在MySQL 8.0.11中添加。
- `keyring_okv_conf_dir`：在MySQL 8.0.11中添加。

- `keyring_operations`：在MySQL 8.0.4中添加。
- `log_error_filter_rules`：在MySQL 8.0.2中添加。
- `log_error_services`：在MySQL 8.0.2中添加。
- `log_error_suppression_list`：在MySQL 8.0.13中添加。
- `mandatory_roles`：在MySQL 8.0.2中添加。
- `mysql_firewall_mode`：在MySQL 8.0.11中添加。
- `mysql_firewall_trace`：在MySQL 8.0.11中添加。
- `mysqlx`：在MySQL 8.0.11中添加。
- `mysqlx-interactive-timeout`：在MySQL 8.0.4中添加。
- `mysqlx-port-read-timeout`：在MySQL 8.0.4中添加。
- `mysqlx-wait-timeout`：在MySQL 8.0.4中添加。
- `mysqlx-write-timeout`：在MySQL 8.0.4中添加。
- `mysqlx_interactive_timeout`：在MySQL 8.0.4中添加。
- `mysqlx_read_timeout`：在MySQL 8.0.4中添加。
- `mysqlx_wait_timeout`：在MySQL 8.0.4中添加。
- `mysqlx_write_timeout`：在MySQL 8.0.4中添加。
- `no-dd-upgrade`：在MySQL 8.0.4中添加。
- `no-monitor`：在MySQL 8.0.12中添加。
- `original_commit_timestamp`：在MySQL 8.0.1中添加。
- `password_history`：在MySQL 8.0.3中添加。
- `password_require_current`：在MySQL 8.0.13中添加。
- `password_reuse_interval`：在MySQL 8.0.3中添加。
- `performance_schema_max_digest_sample_age`：在MySQL 8.0.3中添加。
- `persisted_globals_load`：在MySQL 8.0.0中添加。
- `regexp_stack_limit`：在MySQL 8.0.4中添加。

- `regexp_time_limit`：在MySQL 8.0.4中添加。
- `resultset_metadata`：在MySQL 8.0.3中添加。
- `rpl_read_size`：在MySQL 8.0.11中添加。
- `show_create_table_verbosity`：在MySQL 8.0.11中添加。
- `sql_require_primary_key`：在MySQL 8.0.13中添加。
- `ssl_fips_mode`：在MySQL 8.0.11中添加。
- `syseventlog.facility`：在MySQL 8.0.13中添加。
- `syseventlog.include_pid`：在MySQL 8.0.13中添加。
- `syseventlog.tag`：在MySQL 8.0.13中添加。
- `temptable_max_ram`：在MySQL 8.0.2中添加。
- `thread_pool_algorithm`：在MySQL 8.0.11中添加。
- `thread_pool_high_priority_connection`：在MySQL 8.0.11中添加。
- `thread_pool_max_unused_threads`：在MySQL 8.0.11中添加。
- `thread_pool_prio_kickup_timer`：在MySQL 8.0.11中添加。
- `thread_pool_size`：在MySQL 8.0.11中添加。
- `thread_pool_stall_limit`：在MySQL 8.0.11中添加。
- `use_secondary_engine`：在MySQL 8.0.13中添加。
- `validate_password.check_user_name`：在MySQL 8.0.4中添加。
- `validate_password.dictionary_file`：在MySQL 8.0.4中添加。
- `validate_password.dictionary_file_last_parsed`：在MySQL 8.0.4中添加。
- `validate_password.dictionary_file_words_count`：在MySQL 8.0.4中添加。
- `validate_password.length`：在MySQL 8.0.4中添加。
- `validate_password.mixed_case_count`：在MySQL 8.0.4中添加。
- `validate_password.number_count`：在MySQL 8.0.4中添加。
- `validate_password.policy`：在MySQL 8.0.4中添加。

- validate_password.special_char_count：在MySQL 8.0.4中添加。
- version_compile_zlib：在MySQL 8.0.11中添加。
- windowing_use_high_precision：在MySQL 8.0.2中添加。

选项和变量在MySQL 8.0中不推荐使用

MySQL 8.0中不推荐使用以下系统变量，状态变量和选项。

- expire_logs_days：在这么多天之后清除二进制日志。从MySQL 8.0.3开始不推荐使用。
- innodb_undo_tablespaces：回滚段的表空间文件的数量在两者之间划分。从MySQL 8.0.4开始不推荐使用。
- log_syslog：是否将错误日志写入syslog。从MySQL 8.0.2开始不推荐使用。
- symbolic-links：允许MyISAM表的符号链接。从MySQL 8.0.2开始不推荐使用。

MySQL 8.0中删除的选项和变量

MySQL 8.0中删除了以下系统变量，状态变量和选项。

- Com_alter_db_upgrade：ALTER DATABASE计数...升级数据目录名称语句。在MySQL 8.0.0中删除。
- Innodb_available_undo_logs：显示InnoDB回滚段的总数; 与innodb_rollback_segments不同，后者显示活动回滚段的数量。在MySQL 8.0.2中删除。
- Qcache_free_blocks：查询缓存中的可用内存块数。在MySQL 8.0.3中删除。
- Qcache_free_memory：查询缓存的可用内存量。在MySQL 8.0.3中删除。
- Qcache_hits：查询缓存命中数。在MySQL 8.0.3中删除。
- Qcache_inserts：查询缓存插入的数量。在MySQL 8.0.3中删除。
- Qcache_lowmem_prunes：由于缓存中缺少可用内存而从查询缓存中删除的查询数。在MySQL 8.0.3中删除。
- Qcache_not_cached：非缓存查询的数量（由于query_cache_type设置而不可缓存或未缓存）。在MySQL 8.0.3中删除。
- Qcache_queries_in_cache：在查询缓存中注册的查询数。在MySQL 8.0.3中删除。
- Qcache_total_blocks：查询缓存中的块总数。在MySQL 8.0.3中删除。
- Slave_heartbeat_period：从属的复制心跳间隔，以秒为单位。在MySQL 8.0.1中删除。

- `Slave_last_heartbeat`：以TIMESTAMP格式显示收到最新心跳信号的时间。在MySQL 8.0.1中删除。
- `Slave_received_heartbeats`：自上次重置以来复制从站接收的心跳数。在MySQL 8.0.1中删除。
- `Slave_retried_transactions`：自启动以来复制从属SQL线程已重试事务的总次数。在MySQL 8.0.1中删除。
- `Slave_running`：此服务器的状态为复制从属（从属I / O线程状态）。在MySQL 8.0.1中删除。
- `bootstrap`：由mysql安装脚本使用。在MySQL 8.0.0中删除。
- `date_format`：DATE格式（未使用）。在MySQL 8.0.3中删除。
- `datetime_format`：DATETIME / TIMESTAMP格式（未使用）。在MySQL 8.0.3中删除。
- `des-key-file`：从给定文件加载des_encrypt（）和des_encrypt的密钥。在MySQL 8.0.3中删除。
- `group_replication_allow_local_disjoint_gtids_join`：允许当前服务器加入该组，即使该组中没有事务。在MySQL 8.0.4中删除。
- `have_crypt`：crypt（）系统调用的可用性。在MySQL 8.0.3中删除。
- `ignore-builtin-innodb`：忽略内置的InnoDB。在MySQL 8.0.3中删除。
- `ignore-db-dir`：将目录视为非数据库目录。在MySQL 8.0.0中删除。
- `ignore_db_dirs`：目录被视为非数据库目录。在MySQL 8.0.0中删除。
- `innodb_checksums`：启用InnoDB校验和验证。在MySQL 8.0.0中删除。
- `innodb_disable_resize_buffer_pool_debug`：禁用InnoDB缓冲池的大小调整。在MySQL 8.0.0中删除。
- `innodb_file_format`：新InnoDB表的格式。在MySQL 8.0.0中删除。
- `innodb_file_format_check`：InnoDB是否执行文件格式兼容性检查。在MySQL 8.0.0中删除。
- `innodb_file_format_max`：共享表空间中的文件格式标记。在MySQL 8.0.0中删除。
- `innodb_large_prefix`：为列前缀索引启用更长的键。在MySQL 8.0.0中删除。
- `innodb_locks_unsafe_for_binlog`：强制InnoDB不使用下一键锁定。而是仅使用行级锁定。在MySQL 8.0.0中删除。
- `innodb_stats_sample_pages`：要为索引分布统计信息进行采样的索引页数。在MySQL 8.0.0中删除。
- `innodb_support_xa`：为XA两阶段提交启用InnoDB支持。在MySQL 8.0.0中删除。

- `innodb_undo_logs`：定义InnoDB使用的撤消日志（回滚段）的数量; innodb_rollback_segments的别名。在MySQL 8.0.2中删除。
- `log-warnings`：将一些非关键警告记录到日志文件中。在MySQL 8.0.3中删除。
- `log_builtin_as_identified_by_password`：是否以向后兼容的方式记录CREATE / ALTER USER，GRANT。在MySQL 8.0.11中删除。
- `log_error_filter_rules`：筛选错误日志记录的规则。在MySQL 8.0.4中删除。
- `log_syslog`：是否将错误日志写入syslog。在MySQL 8.0.13中删除。
- `log_syslog_facility`：系统日志消息的工具。在MySQL 8.0.13中删除。
- `log_syslog_include_pid`：是否在syslog消息中包含服务器PID。在MySQL 8.0.13中删除。
- `log_syslog_tag`：标记syslog消息中的服务器标识符。在MySQL 8.0.13中删除。
- `max_tmp_tables`： 没用过。在MySQL 8.0.3中删除。
- `metadata_locks_cache_size`：元数据的大小锁定缓存。在MySQL 8.0.13中删除。
- `metadata_locks_hash_instances`：元数据锁定哈希值的数量。在MySQL 8.0.13中删除。
- `multi_range_count`：范围选择期间一次发送到表处理程序的最大范围数。在MySQL 8.0.3中删除。
- `old_passwords`：为PASSWORD（）选择密码哈希方法。在MySQL 8.0.11中删除。
- `partition`：启用（或禁用）分区支持。在MySQL 8.0.0中删除。
- `query_cache_limit`：不要缓存大于此的结果。在MySQL 8.0.3中删除。
- `query_cache_min_res_unit`：分配结果空间的最小单位大小（在写入所有结果数据后将修剪最后一个单位）。在MySQL 8.0.3中删除。
- `query_cache_size`：分配用于存储旧查询结果的内存。在MySQL 8.0.3中删除。
- `query_cache_type`：查询缓存类型。在MySQL 8.0.3中删除。
- `query_cache_wlock_invalidate`：在LOCK上查询缓存中的查询无效以进行写入。在MySQL 8.0.3中删除。
- `secure-auth`：禁止对具有旧（4.1之前）密码的帐户进行身份验证。在MySQL 8.0.3中删除。
- `show_compatibility_56`：SHOW STATUS / VARIABLES的兼容性。在MySQL 8.0.1中删除。
- `skip-partition`：不要启用用户定义的分区。在MySQL 8.0.0中删除。
- `sync_frm`：在创建时将.frm同步到磁盘。默认情况下启用。在MySQL 8.0.0中删除。

- `temp-pool`：使用此选项将导致创建的大多数临时文件使用一小组名称，而不是每个新文件的唯一名称。在MySQL 8.0.1中删除。
- `time_format`：TIME格式（未使用）。在MySQL 8.0.3中删除。
- `tx_isolation`：默认事务隔离级别。在MySQL 8.0.3中删除。
- `tx_read_only`：默认事务访问模式。在MySQL 8.0.3中删除。

1.6 MySQL信息源

本节列出了您可能会发现有用的其他信息的来源，例如MySQL网站，邮件列表，用户论坛和Internet中继聊天。

1.6.1 MySQL网站

MySQL文档的主要网站是 https://dev.mysql.com/doc/。在线和可下载的文档格式可用于MySQL参考手册，MySQL连接器等。

MySQL开发人员提供有关MySQL服务器博客的新功能和即将推出的功能的信息。

1.6.2 MySQL邮件列表

1.6.2.1使用邮件列表的准则

本节介绍MySQL邮件列表，并提供有关如何使用列表的指南。当您订阅邮件列表时，您会收到列表中的所有帖子作为电子邮件。您也可以将自己的问题和答案发送到列表中。

要订阅或取消订阅本节中描述的任何邮件列表，请访问 http://lists.mysql.com/。对于大多数人，您可以选择获取单个邮件的列表的常规版本，或者每天收到一封大邮件的摘要版本。

请*不要*发送有关订阅或取消订阅任何邮件列表的消息，因为此类消息会自动分发给数千名其他用户。

您的本地站点可能有许多MySQL邮件列表的订阅者。如果是这样，该站点可能具有本地邮件列表，以便从`lists.mysql.com`您的站点发送的邮件传播到本地列表。在这种情况下，请与您的系统管理员联系，以添加到本地MySQL列表或从本地MySQL列表中删除。

要使邮件列表的流量转到邮件程序中的单独邮箱，请根据邮件头设置过滤器。您可以使用`List-ID:`或`Delivered-To:`标题来标识列表消息。

MySQL邮件列表如下：

- `announce`

  新版MySQL及相关程序的公告列表。这是一个所有MySQL用户都应订阅的低容量列表。

- `mysql`

  一般MySQL讨论的主要列表。请注意，在更专业的列表中更好地讨论了一些主题。如果您发布到错误的列表，您可能无法得到答案。

- `bugs`

  希望了解自上次发布MySQL以来所报告的问题或希望积极参与错误搜索和修复过程的人员的列表。请参见第1.7节“如何报告错误或问题”。

- `internals`

  处理MySQL代码的人员列表。这也是讨论MySQL开发和发布补丁的论坛。

- `mysqldoc`

  处理MySQL文档的人员列表。

- `benchmarks`

任何对性能问题感兴趣的人的清单。讨论主要集中在数据库性能（不仅限于MySQL）上，还包括更广泛的类别，如内核性能，文件系统，磁盘系统等。

- `packagers`

  有关打包和分发MySQL的讨论列表。这是分发维护者用来交换包装MySQL的想法以及确保MySQL在所有支持的平台和操作系统上看起来和感觉尽可能相似的论坛。

- `java`

  有关MySQL服务器和Java的讨论列表。它主要用于讨论JDBC Driver / J等JDBC驱动程序。

- `win32`

  有关Microsoft操作系统上MySQL软件的所有主题的列表，例如Windows 9x，Me，NT，2000，XP和2003。

- `myodbc`

  有关使用ODBC连接MySQL服务器的所有主题的列表。

- `gui-tools`

  有关MySQL图形用户界面工具（如MySQL Workbench）的所有主题的列表。

- `cluster`

  讨论MySQL Cluster的列表。

- `dotnet`

  讨论MySQL服务器和.NET平台的列表。它主要与MySQL Connector / NET有关。

- `plusplus`

  有关使用C ++ API for MySQL进行编程的所有主题的列表。

- `perl`

  有关Perl支持MySQL的所有主题的列表 `DBD::mysql`。

如果您无法从MySQL邮件列表或论坛中获得问题的答案，则可以选择从Oracle购买支持。这使您直接与MySQL开发人员联系。

以下MySQL邮件列表使用英语以外的语言。这些列表不由Oracle运营。

- `<mysql-france-subscribe@yahoogroups.com>`

  法国邮件列表。

- <list@tinc.net>

  韩国邮件列表。要订阅，请发送电子邮件subscribe mysql your@email.address至此列表。

- <mysql-de-request@lists.4t2.com>

  德国邮件列表。要订阅，请发送电子邮件subscribe mysql-de your@email.address至此列表。您可以在http://www.4t2.com/mysql/找到有关此邮件列表的 信息。

- <mysql-br-request@listas.linkway.com.br>

  葡萄牙邮件列表。要订阅，请发送电子邮件 subscribe mysql-br your@email.address至此列表。

- <mysql-alta@elistas.net>

  西班牙邮件列表。要订阅，请发送电子邮件subscribe mysql your@email.address至此列表。

1.6.2.1使用邮件列表的准则

请勿在启用HTML模式的情况下从浏览器发布邮件。许多用户不使用浏览器阅读邮件。

当您回答发送到邮件列表的问题时，如果您认为您的答案具有广泛的兴趣，您可能希望将其发布到列表中，而不是直接回复询问的个人。尽量使你的回答足够普通，原始海报以外的人可能会从中受益。当您发布到列表时，请确保您的答案不是以前答案的重复。

尝试在回复中总结问题的基本部分。不要觉得有必要引用整个原始信息。

当答案单独发送给您而不是邮件列表时，总结答案并将摘要发送到邮件列表被认为是礼仪，这样其他人可能会从您收到的回复中获益，帮助您解决问题。

1.6.3 MySQL论坛中的MySQL社区支持

http://forums.mysql.com上的 论坛是一个重要的社区资源。许多论坛都可用，分为以下常规类别：

- 移民
- MySQL用法
- MySQL连接器
- 编程语言
- 工具
- 第三方应用程序
- 存储引擎
- MySQL技术
- SQL标准
- 商业

1.6.4 Internet中继聊天（IRC）上的MySQL社区支持

除了各种MySQL邮件列表和论坛，您还可以在Internet Relay Chat（IRC）上找到经验丰富的社区人员。这些是我们目前所知的最佳网络/频道：

freenode（请参阅 http://www.freenode.net/了解服务器）

- `#mysql`主要用于MySQL问题，但欢迎其他数据库和一般SQL问题。关于PHP，Perl或C与MySQL结合的问题也很常见。
- `#workbench` 主要针对MySQL Workbench相关的问题和想法，它也是一个满足MySQL Workbench开发人员的好地方。

1.6.5 MySQL企业版

Oracle以MySQL Enterprise的形式提供技术支持。对于依赖MySQL DBMS进行业务关键型生产应用程序的组织，MySQL Enterprise是一种商业订阅产品，包括：

- MySQL企业服务器
- MySQL企业监控器
- 每月快速更新和季度服务包
- MySQL知识库
- 全天候技术和咨询支持

MySQL Enterprise提供多层，使您可以灵活地选择最符合您需求的服务级别。有关更多信息，请参阅MySQL Enterprise。

1.7如何报告错误或问题

在发布有关问题的错误报告之前，请尝试验证它是否存在错误，并且尚未报告错误：

- 首先在https://dev.mysql.com/doc/上搜索MySQL在线手册 。我们尝试通过解决新发现的问题来经常更新手册，以使手册保持最新。此外，本手册附带的发行说明特别有用，因为新版本很可能包含解决您问题的方法。发行说明可在手册中给出的位置获得。
- 如果您收到SQL语句的解析错误，请仔细检查您的语法。如果您找不到它的错误，您的当前版本的MySQL服务器很可能不支持您正在使用的语法。如果您使用的是当前版本，并且本手册未涵盖您正在使用的语法，则MySQL Server不支持您的语句。

  如果本手册涵盖了您正在使用的语法，但您有较旧版本的MySQL Server，则应检查MySQL更改历史记录以查看语法的实现时间。在这种情况下，您可以选择升级到较新版本的MySQL Server。
- 有关常见问题的解决方案，请参见 第B.5节“问题和常见错误”。
- 搜索http://bugs.mysql.com/上的错误数据库， 查看是否已报告并修复了错误。
- 在http://lists.mysql.com/上搜索MySQL邮件列表存档 。请参见 第1.6.2节“MySQL邮件列表”。
- 您还可以使用http://www.mysql.com/search/搜索位于MySQL网站的所有网页（包括手册）。

如果您在手册，错误数据库或邮件列表存档中找不到答案，请咨询您当地的MySQL专家。如果您仍无法找到问题的答案，请使用以下指南报告错误。

报告错误的常规方法是访问 http://bugs.mysql.com/，这是我们的错误数据库的地址。该数据库是公开的，任何人都可以浏览和搜索。如果您登录系统，则可以输入新报告。

发布说明中记录了在http://bugs.mysql.com/的错误数据库中发布的针对给定版本进行了更正的错误 。

如果您在MySQL服务器中发现敏感的安全漏洞，请立即通过发送电子邮件告知我们 。例外：支持客户应在http://support.oracle.com/上向Oracle支持部门报告所有问题，包括安全漏洞。
`<secalert_us@oracle.com>`

要讨论其他用户的问题，您可以使用其中一个MySQL邮件列表。第1.6.2节“MySQL邮件列表”。

编写好的错误报告需要耐心，但是第一次正确的做法可以为我们和自己节省时间。一个好的错误报告，包含bug的完整测试用例，使我们很有可能在下一个版本中修复错误。本节可以帮助您正确地编写报告，这样您就不会浪费时间做一些可能对我们没有多大帮助的事情。请仔细阅读本节内容，并确保此处所述的所有信息都包含在您的报告中。

最好在发布之前使用最新的MySQL服务器生产版或开发版测试问题。任何人都应该能够通过使用`mysql test < script_file`您的测试用例或运行您在错误报告中包含的shell或Perl脚本来重复该错误。我们能够重复的任何错误很有可能在下一个MySQL版本中被修复。

当错误报告中包含对问题的良好描述时，它会非常有用。也就是说，举一个很好的例子来说明你所做的一切导致问题并详细描述问题本身。最好的报告包括一个完整的例子，显示如何重现错误或问题。请参见 第28.5节“调试和移植MySQL”。

请记住，我们可以回复包含太多信息的报告，但不能回复包含太少信息的报告。人们经常忽略事实，因为他们认为他们知道问题的原因并假设某些细节无关紧要。一个好的原则是，如果你对陈述某事有疑问，请说出来。如果我们必须要求您提供初始报告中缺少的信息，那么在报告中写几行更快，更麻烦，而不是等待更长时间的答案。

错误报告中最常见的错误是（a）不包括您使用的MySQL发行版的版本号，以及（b）没有完全描述安装MySQL服务器的平台（包括平台类型和版本号）。这些是高度相关的信息，在100个中的99个案例中，如果没有它们，错误报告就毫无用处。我们经常会遇到这样的问题，“为什么这对我不起作用？”然后我们发现请求的功能没有在MySQL版本中实现，或者报告中描述的错误已在较新的MySQL版本中得到修复。错误通常与平台有关。在这种情况下，我们几乎不可能在不知道操作系统和平台版本号的情况下修复任何问题。

如果您从源代码编译MySQL，请记住还提供有关编译器的信息，如果它与问题有关。通常人们会在编译器中发现错误，并认为问题与MySQL有关。大多数编译器都在不断开发中，并且版本越来越好。要确定您的问题是否取决于您的编译器，我们需要知道您使用的编译器。请注意，每个编译问题都应视为错误并进行相应报告。

如果程序产生错误消息，则在报告中包含该消息非常重要。如果我们尝试从档案中搜索某些内容，则报告的错误消息最好与程序生成的错误消息匹配。（甚至应该观察字母。）最好将整个错误消息复制并粘贴到报告中。您永远不应该尝试从内存中重现该消息。

如果您遇到Connector / ODBC（MyODBC）问题，请尝试生成跟踪文件并将其与报告一起发送。请参见 如何报告连接器/ ODBC问题或错误。

如果您的报告包含使用**mysql**命令行工具运行的测试用例的长查询输出行，则可以使用`--vertical`选项或`\G`语句终止符使输出更具可读性。`EXPLAIN SELECT` 本节后面的 示例演示了如何使用 `\G`。

请在报告中包含以下信息：

- 您正在使用的MySQL发行版的版本号（例如，MySQL 5.7.10）。您可以通过执行**mysqladmin版本**找到您正在运行的**版本**。该 **中mysqladmin**程序可以在找到`bin`你的MySQL安装目录下的目录。
- 您遇到问题的机器的制造商和型号。
- 操作系统名称和版本。如果使用Windows，通常可以通过双击“我的电脑”图标并下拉“ 帮助/关于Windows ”菜单来获取名称和版本号。对于大多数类Unix操作系统，您可以通过执行命令获取此信息`uname -a`。
- 有时，内存量（真实和虚拟）是相关的。如有疑问，请包含这些值。
- `docs/INFO_BIN`MySQL安装文件 的内容。此文件包含有关如何配置和编译MySQL的信息。

- 如果您使用的是MySQL软件的源代码分发，请包含您使用的编译器的名称和版本号。如果您有二进制分发，请包含分发名称。
- 如果在编译期间出现问题，请在发生错误的文件中包含确切的错误消息以及围绕违规代码的几行上下文。
- 如果**mysqld**死了，你还应该报告崩溃**mysqld**的声明。您通常可以通过在启用查询日志记录的情况下运行**mysqld**来获取此信息 ，然后在**mysqld**崩溃后查看日志。请参见 第28.5节“调试和移植MySQL”。
- 如果数据库表与问题相关，请 在错误报告中包含语句的输出。这是获取数据库中任何表的定义的一种非常简单的方法。这些信息有助于我们创建与您所经历的情境相匹配的情境。 `SHOW CREATE TABLE db_name.tbl_name`
- 发生问题时生效的SQL模式可能很重要，因此请报告`sql_mode`系统变量的值 。对于存储过程，存储函数和触发器对象，相关`sql_mode`值是创建对象时生效的值。对于存储过程或函数，`SHOW CREATE PROCEDURE`or `SHOW CREATE FUNCTION`语句显示相关的SQL模式，或者您可以查询`INFORMATION_SCHEMA`信息：

  ```
  SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.ROUTINES;
  ```

  对于触发器，您可以使用以下语句：

  ```
  SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.TRIGGERS;
  ```

- 对于与性能相关的错误或`SELECT`语句问题 ，应始终包括语句的输出`EXPLAIN SELECT ...`，以及该`SELECT`语句产生的行数 。您还应该包含所涉及的每个表的输出。您提供的有关您的情况的信息越多，有人可以帮助您的可能性就越大。 `SHOW CREATE TABLE tbl_name`

  以下是一个非常好的错误报告的示例。语句使用**mysql** 命令行工具运行。请注意`\G` 语句终止符用于语句，否则这些语句将提供难以阅读的非常长的输出行。

  ```
  mysql> SHOW VARIABLES;
  mysql> SHOW COLUMNS FROM ...\G
         <output from SHOW COLUMNS>
  mysql> EXPLAIN SELECT ...\G
         <output from EXPLAIN>
  mysql> FLUSH STATUS;
  mysql> SELECT ...;
         <A short version of the output from SELECT,
         including the time taken to run the query>
  mysql> SHOW STATUS;
         <output from SHOW STATUS>
  ```

- 如果在运行**mysqld**时出现错误或问题 ，请尝试提供再现异常的输入脚本。此脚本应包含任何必需的源文件。脚本越能重现您的情况越好。如果您可以制作可重现的测试用例，则应将其上载以附加到错误报告中。

  如果无法提供脚本，则至少应在报告中包含**mysqladmin variables extended-status processlist**的输出，以提供有关系统性能的一些信息。

- 如果您不能生成只有几行的测试用例，或者测试表太大而无法包含在错误报告中（超过10行），则应使用**mysqldump**转储表 并创建`README`描述问题的 文件。使用**tar**和**gzip**或 **zip**创建文件的压缩存档 。在 http://bugs.mysql.com/上为我们的错误数据库启动错误报告后，单击错误报告中的“文件”选项卡，获取有关将存档上载到错误数据库的说明。

- 如果您认为MySQL服务器从语句中产生奇怪的结果，不仅包括结果，还包括您对结果应该是什么的看法，以及描述您的意见基础的解释。

- 当您提供问题的示例时，最好使用实际情况中存在的表名，变量名等，而不是提出新名称。问题可能与表或变量的名称有关。这些情况很少见，但最好是安全而不是抱歉。毕竟，你应该更容易提供一个使用你的实际情况的例子，这对我们来说更好。如果您有错误报告中不希望其他人看到的数据，您可以使用“文件”选项卡上传它，如前所述。如果这些信息真的是绝密的，你甚至不想向我们展示，请继续使用其他名称提供一个例子，

- 如果可能，包括给予相关计划的所有选项。例如，指出启动**mysqld**服务器时使用的选项，以及用于运行任何MySQL客户端程序的选项。**mysqld**和 **mysql**等程序的选项以及 **configure**脚本，通常是解决问题的关键，并且非常重要。包含它们绝不是一个坏主意。如果您的问题涉及用Perl或PHP等语言编写的程序，请包括语言处理器的版本号，以及该程序使用的任何模块的版本。例如，如果您有一个使用`DBI`和`DBD::mysql`模块的Perl脚本，请包含Perl的版本号`DBI`，和 `DBD::mysql`。

- 如果您的问题与权限系统有关，请包含**mysqladmin reload**的输出以及尝试连接时获得的所有错误消息。当您测试您的权限时，您应该执行**mysqladmin reload version**并尝试连接给您带来麻烦的程序。

- 如果你有一个bug补丁，请包含它。但是，如果您没有提供一些必要的信息，例如显示修补程序修复的错误的测试用例，请不要认为补丁是我们所需要的，或者我们可以使用它。我们可能会发现您的补丁有问题，或者我们可能根本不理解它。如果是这样，我们就无法使用它。

  如果我们无法验证补丁的确切用途，我们将不会使用它。测试用例可以帮助我们。显示修补程序处理可能发生的所有情况。如果我们发现一个边界情况（即使是罕见的情况），补丁将不起作用，它可能是无用的。

- 猜测错误是什么，它发生的原因或它依赖的东西通常是错误的。即使MySQL团队在没有首先使用调试器来确定错误的真正原因的情况下也无法猜测这些事情。

- 在您的错误报告中指出您已检查参考手册和邮件存档，以便其他人知道您已尝试自己解决问题。

- 如果您的数据显示已损坏或在访问特定表时出现错误，请首先检查您的表 `CHECK TABLE`。如果该语句报告任何错误：

  - 该`InnoDB`故障恢复机制处理清理服务器时被杀害后重新开始，所以在典型的操作没有必要 “ 修复 ”表。如果遇到`InnoDB`表错误 ，请重新启动服务器并查看问题是否仍然存在，或者错误是否仅影响内存中的缓存数据。如果磁盘上的数据已损坏，请考虑在`innodb_force_recovery` 启用该选项的情况下重新启动， 以便您可以转储受影响的表。

  - 对于非事务性表，请尝试`REPAIR TABLE`使用**myisamchk**或使用 **myisamchk**进行修复 。请参见第5章，*MySQL服务器管理*。

  如果您运行的是Windows，请验证`lower_case_table_names`使用该`SHOW VARIABLES LIKE 'lower_case_table_names'`语句的值 。此变量影响服务器处理数据库和表名称的字母大小的方式。它对给定值的影响应如 第9.2.2节“标识符区分大小写”中所述。

- 如果您经常遇到损坏的表，您应该尝试找出发生这种情况的时间和原因。在这种情况下，MySQL数据目录中的错误日志可能包含有关发生的事情的一些信息。（ 这是`.err` 名称中带有后缀的文件。 ）请参见第5.4.2节“错误日志”。请在错误报告中包含此文件中的所有相关信息。通常情况下**的mysqld**应该 *从来没有*崩溃的表，如果没有在更新过程中把它打死了。如果你能找到**mysqld**死亡的原因， 我们就可以更容易地为你解决这个问题。看到 第B.5.1节“如何确定导致问题的原因”。

- 如果可能，请下载并安装最新版本的MySQL Server，并检查它是否能解决您的问题。所有版本的MySQL软件都经过全面测试，应该可以正常运行。我们相信尽可能地使所有内容都向后兼容，并且您应该能够毫无困难地切换MySQL版本。请参见 第2.1.1节“要安装的MySQL版本和分发版本”。

1.8 MySQL标准合规性

本节描述MySQL如何与ANSI / ISO SQL标准相关。MySQL Server有许多SQL标准的扩展，在这里你可以找到它们是什么以及如何使用它们。您还可以找到有关MySQL Server缺少的功能的信息，以及如何解决一些差异。

SQL标准自1986年以来一直在发展，并且存在多个版本。在本手册中，“ SQL-92 ”是指1992年发布的标准。“ SQL：1999 ”，“ SQL：2003 ”，“ SQL：2008 ”和“ SQL：2011 ”是指发布的标准版本相应的年份，最后一个是最新的版本。我们使用短语“ SQL标准 ”或“ 标准SQL ”在任何时候表示当前版本的SQL标准。

我们对该产品的主要目标之一是继续努力遵守SQL标准，但不会牺牲速度或可靠性。我们不怕添加SQL扩展或支持非SQL功能，如果这大大增加了MySQL Server对我们用户群的大部分可用性。该HANDLER接口是这一战略的一个例子。请参见第13.2.4节“HANDLER语法”。

我们继续支持事务性和非事务性数据库，以满足任务关键型24/7使用和繁重的Web或日志记录使用。

MySQL服务器最初设计用于在小型计算机系统上使用中型数据库（1000万-100万行，或每个表约100MB）。今天MySQL服务器处理TB级数据库。

虽然MySQL复制功能提供了重要的功能，但我们并不是以实时支持为目标。

MySQL支持ODBC级别0到3.51。

MySQL使用NDBCLUSTER存储引擎支持高可用性数据库集群 。请参阅 MySQL NDB Cluster 7.5和NDB Cluster 7.6。

我们实现支持大多数W3C XPath标准的XML功能。请参见第12.11节“XML函数”。

MySQL支持RFC 7159定义的本机JSON数据类型，并基于ECMAScript标准（ECMA-262）。请参见 第11.6节“JSON数据类型”。MySQL还实现了SQL：2016标准的预发布草案所指定的SQL / JSON函数的子集; 有关更多信息，请参见第12.16节“JSON函数”。

选择SQL模式

MySQL服务器可以在不同的SQL模式下运行，并且可以针对不同的客户端以不同的方式应用这些模式，具体取决于sql_mode系统变量的值。DBA可以设置全局SQL模式以匹配站点服务器操作要求，并且每个应用程序可以将其会话SQL模式设置为其自己的要求。

模式会影响MySQL支持的SQL语法以及它执行的数据验证检查。这使得在不同环境中使用MySQL以及将MySQL与其他数据库服务器一起使用变得更加容易。

有关设置SQL模式的更多信息，请参见 第5.1.10节“服务器SQL模式”。

在ANSI模式下运行MySQL

要在ANSI模式下运行MySQL Server，请 使用该选项启动**mysqld**`--ansi`。以ANSI模式运行服务器与使用以下选项启动服务器相同：

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

要在运行时实现相同的效果，请执行以下两个语句：

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode = 'ANSI';
```

您可以看到设置 `sql_mode`系统变量以 `'ANSI'`启用与ANSI模式相关的所有SQL模式选项，如下所示：

```
mysql> SET GLOBAL sql_mode='ANSI';
mysql> SELECT @@global.sql_mode;
        -> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

以ANSI模式运行服务器与 `--ansi`设置SQL模式不完全相同，`'ANSI'`因为该 `--ansi`选项还设置了事务隔离级别。

请参见第5.1.6节“服务器命令选项”。

1.8.1标准SQL的MySQL扩展

MySQL Server支持一些您可能在其他SQL DBMS中找不到的扩展。请注意，如果您使用它们，您的代码将无法移植到其他SQL服务器。在某些情况下，您可以使用以下格式的注释编写包含MySQL扩展的代码，但仍可移植。

```
/*! MySQL-specific code */
```

在这种情况下，MySQL Server会像在任何其他SQL语句中一样解析和执行注释中的代码，但其他SQL服务器将忽略这些扩展。例如，MySQL Server `STRAIGHT_JOIN`在以下语句中识别关键字，但其他服务器不会：

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

如果在`!`字符后添加版本号，则仅当MySQL版本大于或等于指定的版本号时，才会执行注释中的语法。`KEY_BLOCK_SIZE`以下注释中的子句仅由MySQL 5.1.10或更高版本的服务器执行：

```
CREATE TABLE t1(a INT, KEY (a)) /*!50110 KEY_BLOCK_SIZE=1024 */;
```

以下描述列出了按类别组织的MySQL扩展。

- 磁盘上的数据组织

  MySQL服务器将每个数据库映射到MySQL数据目录下的目录，并将数据库中的表映射到数据库目录中的文件名。因此，在具有区分大小写的文件名（例如大多数Unix系统）的操作系统上，MySQL Server中的数据库和表名称区分大小写。请参见 第9.2.2节“标识符区分大小写”。

- 通用语言语法

  - 默认情况下，字符串可以通过封闭`"`和`'`。如果`ANSI_QUOTES`启用了SQL模式，则只能用`'`字符串括起字符串，服务器会解释由`"`标识符括起来的字符串。
  - `\`是字符串中的转义字符。
  - 在SQL语句中，您可以使用***`db_name.tbl_name`***语法访问不同数据库中的表 。一些SQL服务器提供相同的功能，但调用它`User space`。MySQL服务器不支持像这样的语句中使用的表空间：`CREATE TABLE ralph.my_table ... IN my_tablespace`。

- SQL语句语法

  - 在`ANALYZE TABLE`， `CHECK TABLE`， `OPTIMIZE TABLE`，和 `REPAIR TABLE`语句。

- 的CREATE DATABASE， DROP DATABASE和 ALTER DATABASE 语句。请参见第13.1.11节“CREATE DATABASE语法”， 第13.1.22节“DROP DATABASE语法”和 第13.1.2节“ALTER DATABASE语法”。
- 该DO声明。
- EXPLAIN SELECT 获取查询优化器如何处理表的描述。
- 该FLUSH和 RESET语句。
- 该 SET 声明。请参见第13.7.5.1节“变量赋值的SET语法”。
- 该SHOW声明。请参见 第13.7.6节“显示语法”。许多MySQL特定SHOW语句产生的信息 可以通过使用SELECT查询 以更标准的方式获得 INFORMATION_SCHEMA。请参见 第24章， *INFORMATION_SCHEMA表*。
- 使用LOAD DATA INFILE。在许多情况下，此语法与Oracle兼容 LOAD DATA INFILE。请参见第13.2.7节“LOAD DATA INFILE语法”。
- 使用RENAME TABLE。请参见 第13.1.33节“RENAME TABLE语法”。
- 使用REPLACE而不是 DELETE加号 INSERT。请参见 第13.2.9节“REPLACE语法”。
- 使用的， 或者 ， 或者 在 声明。使用多个的， ， ，或 在条款 声明。请参见第13.1.8节“ALTER TABLE语法”。 CHANGE ***col_name***DROP ***col_name***DROP INDEXIGNORERENAMEALTER TABLEADDALTERDROPCHANGEALTER TABLE
- 索引名，索引的列上的前缀，并使用用途INDEX或 KEY在CREATE TABLE声明。请参见 第13.1.18节“CREATE TABLE语法”。
- 使用TEMPORARY或IF NOT EXISTS与CREATE TABLE。
- 使用IF EXISTS与 DROP TABLE和 DROP DATABASE。
- 使用单个DROP TABLE语句删除多个表的功能 。
- 和 语句 的ORDER BY和 LIMIT条款 。 UPDATEDELETE
- INSERT INTO ***tbl_name*** SET ***col_name*** = ... 句法。
- 和 陈述 的DELAYED条款 。 INSERTREPLACE
- 在LOW_PRIORITY该条款 INSERT， REPLACE， DELETE，和 UPDATE语句。
- 使用INTO OUTFILE或INTO DUMPFILE在 SELECT声明中。请参见 第13.2.10节“SELECT语法”。
- 诸如STRAIGHT_JOIN或 SQL_SMALL_RESULT在 SELECT陈述中的选项。

- 您不需要在GROUP BY子句中命名所有选定的列 。这为一些非常具体但非常正常的查询提供了更好的性能。请参见 第12.19节“聚合（GROUP BY）函数”。
- 您可以指定ASC和 DESC使用GROUP BY，而不仅仅是ORDER BY。
- 能够使用:=赋值运算符在语句中设置变量 。请参见 第9.4节“用户定义的变量”。

- 数据类型
  - 的MEDIUMINT， SET和 ENUM数据类型，以及各种BLOB和 TEXT数据类型。
  - 的AUTO_INCREMENT， BINARY，NULL， UNSIGNED，和 ZEROFILL数据类型属性。

- 功能和操作员
  - 为了使从其他SQL环境迁移的用户更容易，MySQL Server支持许多功能的别名。例如，所有字符串函数都支持标准SQL语法和ODBC语法。
  - MySQL服务器理解 ||和 && 运营商当作逻辑OR和AND，如在C编程语言。在MySQL服务器中， || 并且 OR是同义词，因为 && 和AND。由于这个很好的语法，MySQL Server不支持||字符串连接的标准SQL 运算符; 使用 CONCAT()来代替。因为 CONCAT()需要任意数量的参数，所以很容易将||运算符的使用转换 为MySQL服务器。
  - 使用where 有多个元素。 COUNT(DISTINCT ***value_list***)***value_list***
  - 默认情况下，字符串比较不区分大小写，排序顺序由当前字符集的排序规则确定utf8mb4，默认情况下。要执行区分大小写的比较，您应该BINARY使用属性声明列 或使用 BINARY强制转换，这会导致使用基础字符代码值而不是词法排序进行比较。
  - 该% 运营商的代名词 MOD()。也就是说， 相当于 。 C程序员支持并与PostgreSQL兼容。 ***N*** % ***M***MOD(***N***,***M***)%
  - 的=， <>， <=， <， >=， >， <<， >>， <=>， AND， OR，或 LIKE 操作者可以在输出列列表中的表达式中使用（到的左侧FROM）的SELECT语句。例如：

```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```

  - 该LAST_INSERT_ID() 函数返回最新 AUTO_INCREMENT值。请参见 第12.14节“信息功能”。
  - LIKE 允许使用数值。
  - 的REGEXP和 NOT REGEXP扩展正则表达式运算符。
  - CONCAT()或者 CHAR()使用一个参数或两个以上的参数。（在MySQL Server中，这些函数可以使用可变数量的参数。）

- 的BIT_COUNT()，CASE，ELT()，FROM_DAYS()，FORMAT()，IF()，MD5()，PERIOD_ADD()，PERIOD_DIFF()，TO_DAYS()，和WEEKDAY()功能。
- 使用的TRIM()修剪子。标准SQL仅支持删除单个字符。
- 该GROUP BY功能STD()，BIT_OR()，BIT_AND()，BIT_XOR()，和GROUP_CONCAT()。请参见第12.19节“聚合（GROUP BY）函数”。

1.8.2 MySQL与标准SQL的区别

我们尝试使MySQL Server遵循ANSI SQL标准和ODBC SQL标准，但MySQL Server在某些情况下执行不同的操作：

- MySQL和标准SQL权限系统之间存在一些差异。例如，在MySQL中，删除表时不会自动撤消表的权限。您必须显式发出 `REVOKE`语句以撤消表的权限。有关更多信息，请参见 第13.7.1.8节“REVOKE语法”。
- 该`CAST()`函数不支持强制转换为`REAL`或 `BIGINT`。请参见 第12.10节“强制转换函数和运算符”。

1.8.2.1 SELECT INTO TABLE差异

MySQL Server不支持`SELECT ... INTO TABLE`Sybase SQL扩展。相反，MySQL Server支持 `INSERT INTO ... SELECT`标准的SQL语法，这基本上是相同的。请参见第13.2.6.1节“INSERT ... SELECT语法”。例如：

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

或者，您可以使用 `SELECT ... INTO OUTFILE`或 `CREATE TABLE ... SELECT`。

您可以使用`SELECT ... INTO`用户定义的变量。在使用游标和局部变量的存储例程中也可以使用相同的语法。请参见第13.2.10.1节“SELECT ... INTO语法”。

1.8.2.2更新差异

如果从表中访问要在表达式中更新UPDATE的列，请使用列的当前值。以下语句中的第二个赋值设置col2为当前（更新的）col1值，而不是原始col1值。结果就是 col1并且col2具有相同的值。此行为与标准SQL不同。

```
1  UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

1.8.2.3外键差异

外键的MySQL实现在以下关键方面与SQL标准不同：

- 如果父表中有多个行具有相同的引用键值，则 InnoDB在外键检查中执行操作，就好像具有相同键值的其他父行不存在一样。例如，如果已定义 RESTRICT类型约束，并且存在具有多个父行的子行， InnoDB则不允许删除任何这些父行。

  InnoDB 基于对应于外键约束的索引中的记录，通过深度优先算法执行级联操作。

- 一个FOREIGN KEY引用了非约束UNIQUE关键不是标准的SQL，而是一个 InnoDB扩展。

- 如果ON UPDATE CASCADE或者ON UPDATE SET NULLrecurses更新它在同一个级联中之前更新的 *同一个表*，它就像是一样 RESTRICT。这意味着您不能使用自我引用ON UPDATE CASCADE 或ON UPDATE SET NULL操作。这是为了防止级联更新导致的无限循环。自引用的ON DELETE SET NULL，在另一方面，是可能的，因为是自引用ON DELETE CASCADE。级联操作可能不会嵌套超过15级。

- 在插入，删除或更新许多行的SQL语句中，逐行检查外键约束（如唯一约束）。执行外键检查时，InnoDB在必须检查的子记录或父记录上设置共享行级锁。MySQL立即检查外键约束; 检查不会延迟到事务提交。根据SQL标准，默认行为应该是延迟检查。也就是说，只有在处理*完整个SQL语句*后才会检查约束。这意味着无法使用外键删除引用自身的行。

有关InnoDB存储引擎如何 处理外键的信息，请参见 第15.8.1.6节“InnoDB和FOREIGN KEY约束”。

1.8.2.4' - '作为评论的开头

标准SQL使用C语法`/* this is a comment */`进行注释，MySQL Server也支持此语法。MySQL还支持对此语法的扩展，使特定于MySQL的SQL能够嵌入到注释中，如第9.6节“注释语法”中所述。

标准SQL使用“ --”作为开始注释序列。MySQL Server `#` 用作开始注释字符。MySQL Server还支持--评论样式的变体。也就是说，--开始 - 注释序列必须后跟空格（或者通过控制字符，例如换行符）。需要该空间来防止使用如下结构的自动生成的SQL查询出现问题，我们会自动插入付款值`payment`：

```
1  UPDATE account SET credit=credit-payment
```

考虑如果`payment`具有负值会发生什么，例如`-1`：

```
1  UPDATE account SET credit=credit--1
```

`credit--1`是SQL中的有效表达式，但 --被解释为注释的开头，表达式的一部分被丢弃。结果是一个与预期完全不同的语句：

```
1  UPDATE account SET credit=credit
```

该声明根本不会产生任何价值变化。这表明允许评论开始 --会产生严重后果。

使用我们的实现需要一个空格， --以便将它识别为MySQL Server中的开始注释序列。因此， `credit--1`使用安全。

另一个安全功能是**mysql** 命令行客户端忽略以--。开头的行 。

1.8.3 MySQL如何处理约束

MySQL使您既可以使用允许回滚的事务表，也可以使用不允许回滚的非事务表。因此，MySQL中的约束处理与其他DBMS略有不同。当您在非事务表中插入或更新了大量行时，我们必须处理这种情况，当发生错误时无法回滚更改。

基本原理是MySQL Server尝试在解析要执行的语句时检测到它可以检测到的任何错误，并尝试从执行语句时发生的任何错误中恢复。我们在大多数情况下这样做，但尚不是全部。

发生错误时MySQL所具有的选项是在中间停止语句或尽可能地从问题中恢复并继续。默认情况下，服务器遵循后一个过程。这意味着，例如，服务器可能将无效值强制转换为最接近的有效值。

有几种SQL模式选项可用于更好地控制错误数据值的处理以及是否继续执行语句或在发生错误时中止。使用这些选项，您可以将MySQL服务器配置为以更传统的方式运行，就像拒绝不正确输入的其他DBMS一样。可以在服务器启动时全局设置SQL模式以影响所有客户端。各个客户端可以在运行时设置SQL模式，这使每个客户端都可以选择最适合其要求的行为。请参见 第5.1.10节“服务器SQL模式”。

以下部分描述了MySQL Server如何处理不同类型的约束。

1.8.3.1 PRIMARY KEY和UNIQUE索引约束

通常，数据更改语句（例如INSERT或 UPDATE）会发生错误，这些语句 会违反主键，唯一键或外键约束。如果您使用的是事务性存储引擎 InnoDB，则MySQL会自动回滚该语句。如果您使用的是非事务性存储引擎，MySQL将停止处理发生错误的行的语句，并使任何剩余的行保持未处理状态。

MySQL支持的IGNORE关键字 INSERT， UPDATE等。如果您使用它，MySQL会忽略主键或唯一键违规，并继续处理下一行。请参阅您正在使用的语句部分（第13.2.6节“INSERT语法”， 第13.2.12节“UPDATE语法”等）。

您可以获取有关使用mysql_info()C API函数实际插入或更新的行数的信息 。您也可以使用该SHOW WARNINGS声明。请参见 第27.7.7.36节“mysql_info（ ）”和 第13.7.6.40节“显示警告语法”。

只有InnoDB表支持外键。请参见第15.8.1.6节“InnoDB和FOREIGN KEY约束”。

1.8.3.2外键约束

外键允许您跨表交叉引用相关数据， 外键约束有助于保持这种展开数据的一致性。

MySQL支持ON UPDATE和ON DELETE外键的引用 CREATE TABLE和 ALTER TABLE声明。可用参照动作RESTRICT（默认）， ，CASCADE，SET NULL和NO ACTION。

SET DEFAULTMySQL服务器也支持，但目前被拒绝为无效 InnoDB。由于MySQL不支持延迟约束检查，NO ACTION因此被视为RESTRICT。有关MySQL支持外键的确切语法，请参见 第13.1.18.6节“使用FOREIGN KEY约束”。

MATCH FULL，MATCH PARTIAL和MATCH SIMPLE被允许，但应避免使用它们，因为它们会导致MySQL服务器忽略同一语句中使用的任何ON DELETE或 ON UPDATE子句。MATCH选项在MySQL中没有任何其他效果，这实际上强制执行MATCH SIMPLE语义全职。

MySQL要求对外键列进行索引; 如果您创建一个具有外键约束但在给定列上没有索引的表，则会创建一个索引。

您可以从INFORMATION_SCHEMA.KEY_COLUMN_USAGE 表中获取有关外键的信息 。此处显示了针对此表的查询示例：

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME
     > FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
     > WHERE REFERENCED_TABLE_SCHEMA IS NOT NULL;
+--------------+---------------+-------------+-----------------+
| TABLE_SCHEMA | TABLE_NAME    | COLUMN_NAME | CONSTRAINT_NAME |
+--------------+---------------+-------------+-----------------+
| fk1          | myuser        | myuser_id   | f               |
| fk1          | product_order | customer_id | f2              |
| fk1          | product_order | product_id  | f1              |
+--------------+---------------+-------------+-----------------+
3 rows in set (0.01 sec)
```

有关InnoDB 表上外键的信息也可以在数据库中的INNODB_FOREIGN和 INNODB_FOREIGN_COLS表中找到INFORMATION_SCHEMA。

只有InnoDB表支持外键。有关外键支持的信息 ，请参见第15.8.1.6节“InnoDB和FOREIGN KEY约束”InnoDB。

1.8.3.3对无效数据的约束

默认情况下，MySQL可以容忍无效或不正确的数据值，并将它们强制为有效的数据输入值。但是，您可以启用严格SQL模式以选择更坏的值的更传统处理，以便服务器拒绝它们并中止它们发生的语句。请参见 第5.1.10节“服务器SQL模式”。

本节介绍MySQL的默认（宽容）行为，以及严格的SQL模式及其区别。

如果您没有使用严格模式，那么无论何时向列中插入“ 不正确 ”值，例如 `NULL`将`NOT NULL` 列插入到列中或将过大的数值插入数字列，MySQL都会将列设置为“ 最佳可能值 ”而不是产生错误：以下规则更详细地描述了它的工作原理：

- 如果您尝试将超出范围的值存储到数字列中，则MySQL Server会存储零，最小可能值或最大可能值，以最接近无效值为准。
- 对于字符串，MySQL存储空字符串或尽可能多的存储在列中的字符串。
- 如果您尝试将不以数字开头的字符串存储到数字列中，MySQL Server将存储0。
- 如第1.8.3.4节“ENUM和SET约束”中所述，处理`ENUM`和 `SET`列的 值无效。
- MySQL的允许你存储某些不正确的日期值插入`DATE`和 `DATETIME`列（ 如 `'2000-02-31'`或 `'2000-02-00'`）。在这种情况下，当应用程序未启用严格的SQL模式时，应用程序将在存储它们之前验证日期。如果MySQL可以存储日期值并检索完全相同的值，MySQL会将其存储为给定值。如果日期完全错误（服务器存储它的能力之外），则特殊的“ 零 ”日期值 `'0000-00-00'`将存储在列中。
- 如果尝试存储`NULL`到不带`NULL`值的列，则单行`INSERT`语句会发生错误 。对于多行`INSERT` 语句或`INSERT INTO ... SELECT`语句，MySQL Server存储列数据类型的隐式默认值。通常，这适用`0`于数字类型，字符串类型为空字符串（ `''`），日期和时间类型为“ 零 ”值。第11.7节“数据类型默认值”中讨论了隐式默认值 。
- 如果`INSERT`语句没有为列指定值，则在列定义包含explicit `DEFAULT`子句时，MySQL会插入其默认值 。如果定义没有这样的`DEFAULT`子句，MySQL会插入列数据类型的隐式默认值。

在非严格模式下使用前面的规则的原因是我们无法在语句开始执行之前检查这些条件。如果我们在更新几行后遇到问题，我们不能回滚，因为存储引擎可能不支持回滚。终止声明的选择并不是那么好; 在这种情况下，更新将“ 完成一半 ”，这可能是最糟糕的情况。在这种情况下，最好“ 尽你所能 ”，然后继续，好像什么也没发生。

您可以使用`STRICT_TRANS_TABLES`或 `STRICT_ALL_TABLES`SQL模式选择更严格的输入值处理 ：

```
SET sql_mode = 'STRICT_TRANS_TABLES';
SET sql_mode = 'STRICT_ALL_TABLES';
```

STRICT_TRANS_TABLES为事务存储引擎启用严格模式，在某种程度上启用非事务性引擎。它的工作原理如下：

- 对于事务存储引擎，语句中任何位置出现的错误数据值都会导致语句中止和回滚。
- 对于非事务性存储引擎，如果要插入或更新的第一行中发生错误，则语句将中止。（当第一行发生错误时，语句可以被中止以保持表不变，就像事务表一样。）第一行之后的行中的错误不会中止语句，因为表已经被更改了第一排。相反，调整错误的数据值并导致警告而不是错误。换句话说，用STRICT_TRANS_TABLES，如果可以在不更改表的情况下完成，则错误的值会导致MySQL回滚到目前为止所做的所有更新。但是一旦表格被更改，进一步的错误会导致调整和警告。

要进行更严格的检查，请启用 STRICT_ALL_TABLES。这与STRICT_TRANS_TABLES除了非事务性存储引擎之外的情况相同， 即使对于第一行之后的行中的错误数据，错误也会中止该语句。这意味着如果在非事务表的多行插入或更新中途发生错误，则会产生部分更新。插入或更新较早的行，但是从错误点开始的行不是。要避免非事务性表，请使用单行语句或使用 STRICT_TRANS_TABLES如果转换警告而不是错误是可以接受的。为避免出现问题，请不要使用MySQL来检查列内容。让应用程序确保它只将有效值传递给数据库是最安全的（并且通常更快）。

无论使用哪种严格的模式选项，您可以通过使用引起的警告被视为错误 INSERT IGNORE或者UPDATE IGNORE，而不是INSERT或 UPDATE不 IGNORE。

1.8.3.4 ENUM和SET约束

ENUM和 SET列提供了一种有效的方法来定义只能包含给定值集的列。请参见第11.4.4节“ENUM类型”和 第11.4.5节“SET类型”。

如果启用了严格模式（请参见第5.1.10节“服务器SQL模式”），则ENUM或 SET列的定义将作为对输入到列中的值的约束。对于不满足以下条件的值，会发生错误：

- ENUM值必须是列定义中列出 的值之一，或其内部数值等效值。该值不能是错误值（即0或空字符串）。对于定义为一列 ENUM('a','b','c')，值，如''，'d'或者 'ax'是无效的，并且将被拒绝。
- 甲SET值必须是空字符串或由仅在由逗号分隔的列定义中列出的值的值。对于定义为的列 SET('a','b','c')，诸如'd'或 'a,b,c,d'无效的值将被拒绝。

如果使用INSERT IGNORE或，可以在严格模式下抑制无效值的错误UPDATE IGNORE。在这种情况下，会生成警告而不是错误。对于 ENUM，该值将作为错误成员（0）插入。对于 SET，除非删除任何无效的子字符串，否则将以给定的值插入值。例如，'a,x,b,y'结果值为 'a,b'。

1.9学分

以下部分列出了帮助MySQL成为今天的开发人员，贡献者和支持者。

第2章安装和升级MySQL

目录

本章介绍如何获取和安装MySQL。该程序的摘要如下，后面的部分提供了详细信息。如果您计划将现有版本的MySQL升级到较新版本而不是首次安装MySQL，请参见 第2.11.1节“升级MySQL”，了解有关升级过程以及升级前应考虑的问题的信息。

如果您有兴趣从其他数据库系统迁移到MySQL，请参见第A.8节“MySQL 8.0常见问题解答：迁移”，其中包含有关迁移问题的一些常见问题的答案。

MySQL的安装通常遵循此处概述的步骤：

1. ***确定MySQL是否在您的平台上运行并受支持。***

   请注意，并非所有平台都适用于运行MySQL，并且并非所有运行MySQL的平台都由Oracle Corporation正式支持。有关官方支持的平台的信息，请参阅MySQL网站上的https://www.mysql.com/support/supportedplatforms/database.html。

2. ***选择要安装的分发版。***

   有几个版本的MySQL可用，大多数都有几种分发格式。您可以选择包含二进制（预编译）程序或源代码的预打包发行版。如有疑问，请使用二进制分发。Oracle还为那些想要查看最新开发和测试新代码的人提供了对MySQL源代码的访问。要确定应使用的版本和分发类型，请参见第2.1.1节“要安装的MySQL版本和分发版本”。

3. ***下载要安装的发行版。***

   有关说明，请参见第2.1.2节“如何获取MySQL”。要验证分发的完整性，请使用 第2.1.3节“使用MD5校验和或GnuPG验证程序包完整性”中的说明。

4. ***安装发行版。***

   要从二进制发行版安装MySQL，请使用第2.2节“使用通用二进制文件在Unix / Linux上安装MySQL”中的说明。

   要从源代码发行版或当前开发源代码树安装MySQL，请使用 第2.9节“从源代码安装MySQL”中的说明。

5. ***执行任何必要的安装后设置。***

   安装MySQL后，请参见第2.10节“安装后设置和测试” 以获取有关确保MySQL服务器正常工作的信息。另请参阅 第2.10.4节“保护初始MySQL帐户”中提供的信息。本节介绍如何保护初始MySQL `root`用户帐户，*该*帐户在分配*密码*之前*没有密码*。无论您是使用二进制文件还是源代码分发安装MySQL，本节均适用。

6. 如果要运行MySQL基准脚本，必须提供对MySQL的Perl支持。请参见第2.12节“Perl安装说明”。

有关在不同平台和环境中安装MySQL的说明，请参见平台：

- ***Unix，Linux，FreeBSD***

  有关使用通用二进制文件（例如，`.tar.gz`软件包）在大多数Linux和Unix平台上安装MySQL的说明 ，请参见 第2.2节“使用通用二进制文件在Unix / Linux上安装MySQL”。

  有关完全从源代码分发或源代码存储库构建MySQL的信息，请参见 第2.9节“从源代码安装MySQL”

  有关从源代码安装，配置和构建的特定平台帮助，请参阅相应的平台部分：

  - Linux，包括有关分发特定方法的说明，请参见 第2.5节“在Linux上安装MySQL”。
  - IBM AIX，请参见第2.7节“在Solaris上安装MySQL”。
  - FreeBSD，请参见第2.8节“在FreeBSD上安装MySQL”。

- ***微软Windows***

  有关在Microsoft Windows上安装MySQL的说明，请使用MySQL Installer或Zipped二进制文件，请参见第2.3节“在Microsoft Windows上安装MySQL”。

  有关管理MySQL实例的信息，请参见 第2.3.4节“MySQL通告程序”。

  有关使用Microsoft Visual Studio从源代码构建MySQL的详细信息和说明，请参见第2.9节“从源代码 安装MySQL”。

- ***OS X.***

  要在OS X上安装，包括使用二进制包和本机PKG格式，请参见 第2.4节“在macOS上安装MySQL”。

  有关使用OS X Launch Daemon自动启动和停止MySQL的信息，请参见 第2.4.3节“安装和使用MySQL启动守护程序”。

有关MySQL首选项窗格的信息，请参见 第2.4.4节“安装和使用MySQL首选项窗格”。

第3章教程

目录

本章通过展示如何使用**mysql**客户端程序创建和使用简单数据库来提供MySQL的教程介绍。**mysql**（有时称为“ 终端监视器 ”或只是“ 监视器 ”）是一个交互式程序，使您可以连接到MySQL服务器，运行查询和查看结果。 **mysql**也可以在批处理模式下使用：事先将查询放在文件中，然后告诉 **mysql**执行文件的内容。这里介绍了使用**mysql的**两种方法。

要查看**mysql**提供的选项列表，请使用以下`--help`选项调用它：

```
shell> mysql --help
```

本章假设**MySQL的**安装在您的机器上，以及一个MySQL服务器是提供给你可以连接。如果不是这样，请与您的MySQL管理员联系。（如果 *您*是管理员，则需要查阅本手册的相关部分，例如 第5章，*MySQL服务器管理*。 ）

本章介绍了设置和使用数据库的整个过程。如果您只对访问现有数据库感兴趣，则可能需要跳过描述如何创建数据库及其包含的表的部分。

因为本章本质上是教程，所以必须省略许多细节。有关此处所涉及主题的更多信息，请参阅手册的相关章节。

3.1连接和断开服务器

要连接到服务器，通常需要在调用**mysql**时提供MySQL用户名，并且很可能是密码。如果服务器在您登录的计算机以外的计算机上运行，则还需要指定主机名。请与您的管理员联系，以了解您应该使用哪些连接参数进行连接（即，要使用的主机，用户名和密码）。一旦知道了正确的参数，就应该能够像这样连接：

```
shell> mysql -h host -u user -p
Enter password: ********
```

host并 ***user***表示运行MySQL服务器的主机名和MySQL帐户的用户名。替换适合您的设置的值。该`********`代表你的密码; 当**mysql**显示`Enter password:`提示时输入它。

如果可行，您应该看到一些介绍性信息，然后是`mysql>`提示：

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 8.0.15-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

该`mysql>`提示告诉你 **的mysql**准备为你输入SQL语句。

如果您在运行MySQL的同一台计算机上登录，则可以省略主机，只需使用以下命令：

```
shell> mysql -u user -p
```

如果，当您尝试登录时，会收到错误消息，例如 `ERROR 2002（HY000）：无法通过套接字'/tmp/mysql.sock'（2）连接到本地MySQL服务器`，这意味着MySQL服务器守护程序（Unix）或服务（Windows）未运行。请咨询管理员或参阅适用于您的操作系统的第2章“ *安装和升级MySQL* ”一节 。

有关尝试登录时经常遇到的其他问题的帮助，请参见第B.5.2节“使用MySQL程序时的常见错误”。

某些MySQL安装允许用户以匿名（未命名）用户身份连接到本地主机上运行的服务器。如果你的机器上是这种情况，你应该能够通过调用没有任何选项的**mysql**连接到该服务器：

```
shell> mysql
```

成功连接后，您可以通过在提示符下键入`QUIT`（或`\q`）来随时断开连接`mysql>`：

```
mysql> QUIT
Bye
```

在Unix上，您也可以通过按Control + D断开连接。

以下部分中的大多数示例都假定您已连接到服务器。他们通过`mysql>`提示表明了这一点 。

3.2输入查询

确保已连接到服务器，如上一节中所述。这样做本身并不选择任何可以使用的数据库，但这没关系。此时，更重要的是要找到一些关于如何发出查询的信息，而不是直接创建表，将数据加载到它们中以及从中检索数据。本节介绍了输入查询的基本原则，使用您可以尝试的几个查询来熟悉**mysql的** 工作原理。

这是一个简单的查询，要求服务器告诉您它的版本号和当前日期。按照`mysql>`提示输入如下所示输入，然后按Enter键：

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
+-----------+--------------+
| 5.8.0-m17 | 2015-12-21   |
+-----------+--------------+
1 row in set (0.02 sec)
mysql>
```

这个查询说明了几个关于**mysql的**东西 ：

- 查询通常由一个SQL语句后跟一个分号组成。（有一些例外情况，可以省略分号。`QUIT`前面提到过，分号就是其中之一。我们稍后会找到其他分号。）
- 当您发出查询时，**mysql**将其发送到服务器以执行并显示结果，然后打印另一个`mysql>`提示以指示它已准备好进行另一个查询。
- **mysql**以表格形式（行和列）显示查询输出。第一行包含列的标签。以下行是查询结果。通常，列标签是从数据库表中提取的列的名称。如果您正在检索表达式的值而不是表列（如刚刚显示的示例中所示），则 **mysql**使用表达式本身标记列。
- **mysql**显示返回了多少行以及执行查询所需的时间，这使您可以大致了解服务器性能。这些值是不精确的，因为它们代表挂钟时间（不是CPU或机器时间），并且因为它们受到服务器负载和网络延迟等因素的影响。（为简洁起见， 本章其余示例中有时未显示“行中的行 ”行。）

关键字可以输入任何字母。以下查询是等效的：

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

这是另一个查询。它演示了您可以将 **mysql**用作简单的计算器：

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

到目前为止显示的查询是相对较短的单行语句。您甚至可以在一行中输入多个语句。用分号结束每一个：

```
mysql> SELECT VERSION(); SELECT NOW();
+-----------+
| VERSION() |
+-----------+
| 8.0.13    |
+-----------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2018-08-24 00:56:40 |
+---------------------+
1 row in set (0.00 sec)
```

不需要在一行上提供查询，因此需要多行的冗长查询不是问题。 **mysql**通过查找终止分号来确定语句的结束位置，而不是查找输入行的结尾。（换句话说，**mysql** 接受自由格式输入：它收集输入行但在看到分号之前不执行它们。）

这是一个简单的多行语句：

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2018-08-24   |
+---------------+--------------+
```

在这个例子中，请注意提示符从 `mysql>`对`->`您输入一个多行查询的第一行后。这就是**mysql**如何 表明它还没有看到完整的声明并且正在等待其余部分。提示是您的朋友，因为它提供了有价值的反馈。如果您使用该反馈，您始终可以了解**mysql** 正在等待什么。

如果您决定不想执行正在输入的查询，请键入`\c`以下内容取消它 ：

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

在这里，也请注意提示。它会`mysql>`在您键入后切换回来 `\c`，提供反馈以指示**mysql**已准备好进行新查询。

下表显示了您可能看到的每个提示，并总结了它们对**mysql**所处状态的含义 。

提示	含义
`mysql>`	准备好进行新查询
`->`	等待多行查询的下一行
`'>`	等待下一行，等待以单引号开头的字符串的完成（`'`）
`">`	等待下一行，等待以双引号开头的字符串的完成（`"`）
`` `> ``	等待下一行，等待以反引号（`` ` ``）开头的标识符的完成
`/*>`	等待下一行，等待以＃开头的评论完成 `/*`

当您打算在一行上发出查询时，通常会出现多行语句，但忘记了终止分号。在这种情况下，**mysql** 等待更多输入：

```
mysql> SELECT USER()
    ->
```

如果你遇到这种情况（你认为你已经输入了一个语句，但唯一的响应是`->`提示），很可能**mysql**正在等待分号。如果您没有注意到提示告诉您的内容，您可能会在那里坐一会儿，然后才意识到您需要做什么。输入分号以完成语句，**mysql** 执行它：

```
mysql> SELECT USER()
    -> ;
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
```

在`'>`和`">`字符串集合（说，MySQL是在等待一个字符串完成的另一种方式）中的提示信息出现。在MySQL中，您可以编写由任一字符`'`或`"`字符包围的字符串（例如，`'hello'`或`"goodbye"`），并且 **mysql**允许您输入跨越多行的字符串。当您看到`'>`或`">`提示时，表示您输入的行包含以a `'` 或`"`引号字符开头的字符串，但尚未输入终止字符串的匹配引号。这通常表明您无意中遗漏了引号字符。例如：

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

如果输入此SELECT语句，则按**Enter键**并等待结果，没有任何反应。而不是想知道为什么这个查询需要这么长时间，请注意'>提示提供的线索。它告诉你**mysql**希望看到未终止的字符串的其余部分。（您是否在语句中看到错误？字符串'Smith缺少第二个单引号。）

在这一点上，你做什么？最简单的方法是取消查询。但是，您不能只键入\c这种情况，因为**mysql**将其解释为它正在收集的字符串的一部分。相反，输入结束引号字符（所以**mysql**知道你已经完成了字符串），然后输入\c：

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

提示符更改回mysql>，表示**mysql**已准备好进行新查询。

该`>提示类似于 '>和">提示，但表示你已经开始但尚未完成反引号引用的标识符。

重要的是要知道什么是重要的'>， ">和`>提示表示，因为如果你错误地输入一个未终止的字符串，你输入的任何进一步线似乎被忽视 **MySQL的** -包括含线 QUIT。这可能非常令人困惑，特别是如果您在取消当前查询之前不知道需要提供终止引用。

3.3.1创建和选择数据库

如果管理员在设置权限时为您创建数据库，则可以开始使用它。否则，您需要自己创建它：

```
mysql> CREATE DATABASE menagerie;
```

在Unix下，数据库名称是区分大小写的（不像SQL关键字），所以你必须总是指到你的数据库 `menagerie`，而不是 `Menagerie`，`MENAGERIE`或一些其他变种。表名也是如此。（在Windows下，此限制不适用，但您必须在给定查询中使用相同的字母表引用数据库和表。但是，出于各种原因，建议的最佳做法始终是使用在使用时使用的相同字母。数据库已创建。）

> **注意**
>
> 如果您收到错误，例如`ERROR 1044（42000）：在尝试创建数据库时，用户'micah'@'localhost'拒绝访问数据库'menagerie'`，这意味着您的用户帐户没有必要的权限所以。与管理员讨论或参见第6.2节“MySQL访问权限系统”。

创建数据库不会选择它使用; 你必须明确地这样做。要创建`menagerie`当前数据库，请使用以下语句：

```
mysql> USE menagerie
Database changed
```

您的数据库只需创建一次，但每次开始**mysql** 会话时都必须选择它才能使用。您可以通过发出`USE`示例中所示的语句来完成此操作 。或者，您可以在调用**mysql**时在命令行上选择数据库。只需在您可能需要提供的任何连接参数后指定其名称。例如：

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

> **重要**
>
> `menagerie`在刚才显示的命令中 ***不是***您的密码。如果要在`-p`选项后的命令行上提供密码，则 必须在没有中间空间的情况下执行此操作（例如，as ，not as ）。但是，建议不要在命令行中输入密码，因为这样做会使其暴露给在您的计算机上登录的其他用户窥探。 `-p`***`password`***`-p` ***`password`***

> **注意**
>
> 您可以随时查看当前选择使用哪个数据库。 `SELECT DATABASE()`

3.3.2创建表

创建数据库很容易，但此时它是空的，SHOW TABLES告诉你：

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

更难的部分是决定数据库的结构应该是什么：您需要哪些表以及每个表中应该包含哪些列。

您想要一张包含每只宠物记录的表格。这可以称为pet表格，它应该包含每个动物名称的最低限度。因为名称本身不是很有趣，所以该表应包含其他信息。例如，如果您家中有多个人饲养宠物，您可能需要列出每只动物的主人。您可能还想记录一些基本的描述性信息，如物种和性别。

年龄怎么样？这可能是有意义的，但存储在数据库中并不是一件好事。随着时间的推移，年龄会发生变化，这意味着您必须经常更新记录。相反，最好存储固定值，如出生日期。然后，无论何时需要年龄，您都可以将其计算为当前日期和出生日期之间的差异。MySQL提供了进行日期算术的功能，因此这并不困难。存储出生日期而不是年龄也有其他优点：

- 您可以使用数据库执行任务，例如为即将到来的宠物生日生成提醒。（如果您认为这种类型的查询有些愚蠢，请注意，您可能会在业务数据库的上下文中询问相同的问题，以确定您需要在当前一周或一个月内向其发送生日祝福的客户，计算机辅助个人触摸。）
- 您可以计算与当前日期以外的日期相关的年龄。例如，如果您将死亡日期存储在数据库中，则可以轻松计算宠物死亡时的年龄。

您可能会想到在pet表格中有用的其他类型的信息，但到目前为止确定的信息是足够的：名称，所有者，物种，性别，出生和死亡。

使用CREATE TABLE语句指定表的布局：

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

VARCHAR对于，和 列是一个很好的选择 name，因为列值的长度不同。这些列定义中的长度不必全部相同，也不必相同。通常，您可以选择从任何长度到 ，无论似乎是最合理的给你。如果你选择不好而后来需要更长的字段，那么MySQL提供了一个声明。 ownerspecies20165535ALTER TABLE

可以选择几种类型的值来表示动物记录中的性别，例如'm'和 'f'，或者可能'male'和 'female'。最简单的是使用单个字符'm'和'f'。

DATE对于birth和death 列 使用数据类型是一个相当明显的选择。

创建表后，SHOW TABLES应该产生一些输出：

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| pet                 |
+---------------------+
```

要验证您的表是否按预期方式创建，请使用以下DESCRIBE语句：

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

您可以随时使用DESCRIBE，例如，如果您忘记了表中列的名称或它们具有的类型。

有关MySQL数据类型的更多信息，请参见 第11章，*数据类型*。

3.3.3将数据加载到表中

创建表后，您需要填充它。该 LOAD DATA和 INSERT语句是这个有用的。

假设您的宠物记录可以如下所示进行描述。（注意MySQL期望'YYYY-MM-DD'格式的日期;这可能与您习惯的不同。）

名称	所有者	种类	性别	分娩	死亡
蓬松	哈罗德	猫	F	1993年2月4日	
爪	格温	猫	米	1994年3月17日	
巴菲	哈罗德	狗	F	1989年5月13日	
方	班尼	狗	米	1990年8月27日	
鲍泽	黛安	狗	米	1979年8月31日	1995年7月29日
抔	格温	鸟	F	1998年9月11日	
惠斯勒	格温	鸟		1997年12月9日	
瘦	班尼	蛇	米	1996年4月29日	

因为您从一个空表开始，所以填充它的一种简单方法是为每个动物创建一个包含行的文本文件，然后使用单个语句将该文件的内容加载到表中。

您可以创建一个文本文件pet.txt ，每行包含一个记录，其值由制表符分隔，并按照CREATE TABLE语句中列出的顺序给出 。对于缺失值（例如未知性别或仍然生活的动物的死亡日期），您可以使用NULL 值。要在文本文件中表示这些，请使用 \N（反斜杠，大写-N）。例如，惠斯勒鸟的记录看起来像这样（值之间的空格是单个制表符）：

```
Whistler        Gwen    bird    \N      1997-12-09      \N
```

要将文本文件加载pet.txt到 pet表中，请使用以下语句：

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

如果您在Windows上使用编辑器创建该文件 \r\n作为行终止符，则应使用此语句：

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
    -> LINES TERMINATED BY '\r\n';
```

（在运行OS X的Apple计算机上，您可能希望使用 LINES TERMINATED BY '\r'。）

如果需要，可以在LOAD DATA语句中显式指定列值分隔符和行结束标记，但默认值为制表符和换行符。这些语句足以使语句pet.txt正确读取文件。

如果语句失败，则默认情况下，您的MySQL安装可能没有启用本地文件功能。有关如何更改此信息的信息，请参见第6.1.6节“LOAD DATA LOCAL的安全问题”。

如果要一次添加一条新记录，该 INSERT语句很有用。在最简单的形式中，您按照CREATE TABLE语句中列出的列的顺序为每列提供值 。假设黛安得到一只名为“ Puffball ”的新仓鼠。“您可以使用如下INSERT语句添加新记录 ：

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

字符串和日期值在此处指定为带引号的字符串。此外，INSERT您可以NULL直接插入 以表示缺失值。你没有\N像你一样使用 LOAD DATA。

从这个示例中，您应该能够看到，最初使用多个INSERT语句而不是单个LOAD DATA 语句来加载记录会涉及更多类型。

3.3.4.1选择所有数据

最简单的形式SELECT 从表中检索所有内容：

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

SELECT如果要查看整个表格，例如在刚刚加载初始数据集之后， 这种形式非常有用。例如，您可能会认为Bowser的出生日期似乎不太合适。咨询你原来的血统书，你会发现正确的出生年应该是1989年，而不是1979年。

至少有两种方法可以解决这个问题：

- 编辑文件pet.txt以更正错误，然后清空表并使用DELETE和 重新加载它 LOAD DATA：

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
  ```

  但是，如果这样做，您还必须重新输入Puffball的记录。

- 仅使用UPDATE语句修复错误记录 ：

  ```
  mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
  ```

  该UPDATE变化只在讨论记录，并不需要您重新加载表。

3.3.4.2选择特定行

如上一节所示，可以轻松检索整个表。只需省略声明中的`WHERE`条款即可`SELECT`。但通常你不希望看到整个表格，特别是当它变大时。相反，您通常对回答特定问题更感兴趣，在这种情况下，您可以对所需信息指定一些约束。让我们看看他们回答的有关您的宠物的问题的一些选择查询。

您只能从表中选择特定行。例如，如果您想验证您对Bowser出生日期所做的更改，请选择Bowser的记录，如下所示：

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

输出结果证实，这一年被正确记录为1989年，而不是1979年。

字符串比较通常不区分大小写，因此您可以将名称指定为`'bowser'`，`'BOWSER'`等等。查询结果是一样的。

您可以在任何列上指定条件，而不仅仅是`name`。例如，如果您想知道1998年或之后出生的动物，请测试该`birth`栏：

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

例如，您可以结合条件来定位雌性狗：

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

前面的查询使用`AND`逻辑运算符。还有一个`OR`运营商：

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

AND并且 OR可以混合，但 AND优先级高于 OR。如果您同时使用这两个运算符，最好使用括号明确指出条件应如何分组：

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
    -> OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.3选择特定列

如果您不想查看表中的整行，只需将您感兴趣的列命名为逗号分隔。例如，如果您想知道您的动物何时出生，请选择`name`和 `birth`列：

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

要找出谁拥有宠物，请使用此查询：

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

请注意，查询只是`owner`从每条记录中检索 列，其中一些列出现不止一次。要最小化输出，请通过添加关键字检索每个唯一的输出记录一次 `DISTINCT`：

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+

```

```
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

您可以使用`WHERE`子句将行选择与列选择组合在一起。例如，要仅获取狗和猫的出生日期，请使用以下查询：

```
mysql> SELECT name, species, birth FROM pet
    -> WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4排序行

您可能已在前面的示例中注意到结果行没有按特定顺序显示。当行以某种有意义的方式排序时，通常更容易检查查询输出。要对结果进行排序，请使用`ORDER BY`子句。

这是动物的生日，按日期排序：

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

在字符类型列上，排序与所有其他比较操作一样，通常以不区分大小写的方式执行。这意味着除了它们的情况之外，对于相同的列，订单是未定义的。您可以使用如下方式强制对列进行区分大小写的排序`BINARY`：。`ORDER BY BINARY col_name`

默认排序顺序为升序，首先是最小值。要按反向（降序）顺序排序，请将`DESC`关键字添加到要排序 的列的名称：

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

您可以对多个列进行排序，并且可以按不同方向对不同列进行排序。例如，要按动物类型按升序排序，然后按动物类型中的出生日期按降序排序（最年轻的动物首先），请使用以下查询：

```
mysql> SELECT name, species, birth FROM pet
    -> ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

该`DESC`关键字仅适用于紧邻其前面的列名（`birth`）；它不会影响`species`列排序顺序。

3。3。4。5日期计算

MySQL提供了几个可用于执行日期计算的函数，例如，计算年龄或提取日期的部分。

要确定每只宠物的年龄，请使用此 `TIMESTAMPDIFF()` 功能。它的参数是您希望表达结果的单位，以及两个可以区分的日期。以下查询显示每只宠物的出生日期，当前日期和年龄。一个 *别名*（`age`）是用来制造最终输出列标签更有意义。

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

查询有效，但如果以某种顺序显示行，则可以更轻松地扫描结果。这可以通过添加一个`ORDER BY name`子句来按名称对输出进行排序来完成：

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet ORDER BY name;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

要通过age而不是 对输出进行排序name，只需使用不同的ORDER BY子句：

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

类似的查询可用于确定死亡动物的死亡年龄。您可以通过检查death值 是否确定这些动物NULL。然后，对于那些具有非NULL值的人，计算death和 birth值之间的差异：

```
mysql> SELECT name, birth, death,
    -> TIMESTAMPDIFF(YEAR,birth,death) AS age
    -> FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

查询使用death IS NOT NULL而不是death <> NULL因为 NULL是一个特殊值，无法使用通常的比较运算符进行比较。这将在稍后讨论。请参见第3.3.4.6节“使用NULL值”。

如果你想知道哪些动物下个月有生日怎么办？对于这种类型的计算，年和日是无关紧要的; 您只想提取birth列的月份部分 。MySQL提供了用于提取日期的部分，如一些功能 YEAR()， MONTH()和 DAYOFMONTH()。MONTH()这是适当的功能。看看它是如何工作的，运行，显示两者的价值一个简单的查询birth和MONTH(birth)：

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
```

```
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

在接下来的一个月里寻找有生日的动物也很简单。假设当前月份是4月。然后月份值是`4`，你可以找到5月（月`5`）出生的动物，如下所示：

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

如果当前月份是12月，则会出现一个小的复杂情况。你不能只在月份数字（`12`）中添加一个并查找月份出生的动物`13`，因为没有这样的月份。相反，你寻找1月（月`1`）出生的动物 。

您可以编写查询，以便无论当前月份是什么，它都可以工作，因此您不必使用特定月份的数字。`DATE_ADD()`使您可以将时间间隔添加到给定日期。如果您将值添加一个月`CURDATE()`，然后使用，则提取月份部分`MONTH()`，结果将生成查找生日的月份：

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

完成相同任务的另一种方法是`1`在使用modulo函数（`MOD`）将月值包装`0`为当前值之后，添加 以获取当前后一个月之后的下个月`12`：

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

`MONTH()`返回`1`和之间的数字`12`。并 `MOD(something,12)`在`0`和之间返回一个数字`11`。所以添加必须在之后`MOD()`，否则我们将从11月（`11`）到1月（`1`）。

如果计算使用无效日期，则计算失败并生成警告：

```
mysql> SELECT '2018-10-31' + INTERVAL 1 DAY;
+-----------------------------+
| '2018-10-31' + INTERVAL 1 DAY |
```

```
+------------------------------+
| 2018-11-01                   |
+------------------------------+
mysql> SELECT '2018-10-32' + INTERVAL 1 DAY;
+------------------------------+
| '2018-10-32' + INTERVAL 1 DAY |
+------------------------------+
| NULL                         |
+------------------------------+
mysql> SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1292 | Incorrect datetime value: '2018-10-32' |
+---------+------+---------------------------------------+
```

3.3.4.6使用NULL值

在NULL您习惯它之前，这个值可能会令人惊讶。从概念上讲，NULL意味着“缺失的未知值”，并且与其他值的处理方式略有不同。

要测试NULL，请使用 IS NULL和IS NOT NULL运算符，如下所示：

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

你不能使用算术比较操作符，如 =， <或 <>以测试NULL。要自己演示，请尝试以下查询：

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

由于任何算术比较的结果 NULL也是如此NULL，因此您无法从此类比较中获得任何有意义的结果。

在MySQL中，0或NULL 意味着虚假，其他任何意味着真实。布尔运算的默认真值是1。

这种特殊处理NULL是为什么在上一节中，有必要确定哪些动物不再使用death IS NOT NULL而不是death <> NULL。

NULL在a中， 两个值被视为相等 GROUP BY。

在执行操作时ORDER BY，NULL如果您这样做ORDER BY ... ASC，则首先显示值，如果执行， 则显示最后值 ORDER BY ... DESC。

使用时常见的错误NULL是假设无法将零或空字符串插入到定义为的列中NOT NULL，但事实并非如此。这些实际上是价值，而 NULL意味着“ 没有价值。“你可以通过使用IS [NOT] NULL如下所示轻松地测试这个 ：

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----------+---------------+------------+----------------+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----------+---------------+------------+----------------+

```

```
6 |         0 |             1 |          0 |              1 |
  +-----------+---------------+------------+----------------+
```

因此，完全可以将零或空字符串插入到`NOT NULL`列中，因为这些实际上是这样的`NOT NULL`。请参见 第B.5.4.3节“NULL值的问题”。

3.3.4.7模式匹配

MySQL提供标准的SQL模式匹配以及基于类似于Unix实用程序（如**vi**，**grep**和 **sed** ）使用的扩展正则表达式的模式匹配形式 。

SQL模式匹配使您可以使用_ 匹配任何单个字符并%匹配任意数量的字符（包括零个字符）。在MySQL中，SQL模式默认情况下不区分大小写。这里显示了一些例子。不要使用 =或<>使用SQL模式时。请改用LIKE或NOT LIKE比较运算符。

要查找以下开头的名称b：

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

要查找以以下结尾的名称fy：

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

要查找包含名称w：

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

要查找包含五个字符的名称，请使用_模式字符的五个实例：

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

MySQL提供的另一种模式匹配使用扩展的正则表达式。当您测试此类模式的匹配项时，请使用 REGEXP_LIKE()函数（ REGEXP或 RLIKE 运算符，它们是同义词 REGEXP_LIKE()）。

以下列表描述了扩展正则表达式的一些特征：

- . 匹配任何单个字符。
- 字符类[...]匹配括号内的任何字符。例如， [abc]匹配a， b或c。要命名一系列字符，请使用短划线。[a-z] 匹配任何字母，而[0-9] 匹配任何数字。
- *匹配前面的事物的零个或多个实例。例如，x* 匹配任意数量的x字符， [0-9]*匹配任意数量的数字，并.*匹配任意数量的任何数字。
- 如果模式匹配正在测试的值中的任何位置，则正则表达式模式匹配成功。（这与LIKE模式匹配不同，模式匹配仅在模式匹配整个值时才会成功。）
- 锚定的图案，使得它必须在值的开头或结尾匹配正在测试中，使用^在一开始或$在图案的端部。

为了演示扩展正则表达式的工作原理，LIKE先前显示的 查询将在此处重写以供使用 REGEXP_LIKE()。

要查找以...开头的名称b，请使用 ^以匹配名称的开头：

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b');
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

要强制将正则表达式比较区分大小写，请使用区分大小写的排序规则，或使用 BINARY关键字使其中一个字符串成为二进制字符串，或指定c 匹配控制字符。这些查询中的每b一个在名称的开头仅匹配小写：

```
SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b' COLLATE utf8mb4_0900_as_cs);
SELECT * FROM pet WHERE REGEXP_LIKE(name, BINARY '^b');
SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b', 'c');
```

要查找以？结尾的名称`fy`，请使用 `$`以匹配名称的结尾：

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, 'fy$');
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

要查找包含a的名称`w`，请使用以下查询：

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, 'w');
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

因为正则表达式模式匹配，如果它出现在值的任何位置，则在前一个查询中不需要在模式的任何一侧放置通配符以使其与整个值匹配，就像SQL模式一样。

要查找包含五个字符的名称，请使用 `^`和`$`匹配名称的开头和结尾，以及`.`中间的五个实例 ：

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^.....$');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

您还可以使用（“repeat- -times”）运算符编写上一个查询 ：`{`***n***`}`***n***

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^.{5}$');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

有关正则表达式语法的更多信息，请参见第12.5.2节“正则表达式”。

3.3.4.8计数行

数据库通常用于回答“ 表格中某种类型的数据出现频率的问题？“ 例如，您可能想知道您拥有多少只宠物，或每个拥有者拥有多少只宠物，或者您可能想要对您的动物进行各种类型的人口普查操作。

计算你拥有的动物总数与“ 桌子中有多少行 `pet` ”是同一个问题？“因为每只宠物有一条记录。`COUNT(*)`计算行数，因此计算动物的查询如下所示：

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

之前，您检索了拥有宠物的人的姓名。`COUNT()`如果您想了解每位业主拥有多少宠物，您可以使用：

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

上述查询用于`GROUP BY`对每个记录进行分组`owner`。使用的 `COUNT()` 结合 `GROUP BY`是在各种分组表征您的数据非常有用。以下示例显示了执行动物普查操作的不同方法。

每种动物数量：

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

每性别的动物数量：

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

（在此输出中，NULL表示性别未知。）

每种物种和性别组合的动物数量：

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

使用时无需检索整个表 COUNT()。例如，以前的查询，只在狗和猫上执行时，如下所示：

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE species = 'dog' OR species = 'cat'
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

或者，如果你想要每性别的动物数量只对已知性别的动物：

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE sex IS NOT NULL
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

如果除了`COUNT()`值之外还要指定要选择的列 ，则`GROUP BY`应该存在一个用于命名相同列的子句。否则，会发生以下情况：

- 如果`ONLY_FULL_GROUP_BY`启用了 SQL模式，则会发生错误：

```
mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression
#1 of SELECT list contains nonaggregated column 'menagerie.pet.owner';
this is incompatible with sql_mode=only_full_group_by
```

- 如果`ONLY_FULL_GROUP_BY`未启用，则通过将所有行视为单个组来处理查询，但为每个命名列选择的值是不确定的。服务器可以自由选择任何行中的值：

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Harold |        8 |
+--------+----------+
1 row in set (0.00 sec)
```

另请参见第12.19.3节“GROUP BY的MySQL处理”。有关 行为和相关优化的信息 ，请参见 第12.19.1节“聚合（GROUP BY）函数描述”。 `COUNT(expr)`

3.3.4.9使用多个表

该pet表记录了您拥有的宠物。如果您想记录有关它们的其他信息，例如生活中的事件，例如兽医的访问或者出生时，您需要另一张桌子。这张桌子应该是什么样的？它需要包含以下信息：

- 宠物名称，以便您了解每个事件所属的动物。
- 日期，以便您知道事件发生的时间。
- 描述事件的字段。
- 如果您希望能够对事件进行分类，则为事件类型字段。

鉴于这些注意事项， 表的CREATE TABLE语句event可能如下所示：

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

与pet表一样，最简单的方法是通过创建包含以下信息的制表符分隔文本文件来加载初始记录。

名称	日期	类型	备注
蓬松	1995年5月15日	窝	4只小猫，3只雌性，1只雄性
巴菲	1993-06-23	窝	5只小狗，2只雌性，3只雄性
巴菲	1994年6月19日	窝	3只小狗，3只雌性
抅	1999年3月21日	兽医	需要喙拉直
瘦	1997年8月3日	兽医	断肋骨
鲍泽	1991年10月12日	狗窝	
方	1991年10月12日	狗窝	
方	1998年8月28日	生日	给了他一个新的咀嚼玩具
爪	1998年3月17日	生日	给了他一个新的跳蚤领
惠斯勒	1998年12月9日	生日	第一个生日

像这样加载记录：

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

根据您从pet表中运行的查询中学到的内容，您应该能够对event表中的记录执行检索；原则是一样的。但是，event表格本身何时不足以回答您可能会问的问题？

假设你想要找出每只宠物的窝的年龄。我们之前看到过如何计算两个日期的年龄。母亲的垃圾日期在 event 表格中，但要计算她在该日期的年龄，您需要她的出生日期，该日期存储在 pet表格中。这意味着查询需要

两个表：

```
mysql> SELECT pet.name,
    -> TIMESTAMPDIFF(YEAR,birth,date) AS age,
    -> remark
    -> FROM pet INNER JOIN event
    ->   ON pet.name = event.name
    -> WHERE event.type = 'litter';
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

有关此查询的注意事项有以下几点：

- 该`FROM`子句连接两个表，因为查询需要从两个表中提取信息。
- 组合（连接）来自多个表的信息时，需要指定一个表中的记录如何与另一个表中的记录匹配。这很容易，因为它们都有一个`name`列。该查询使用`ON`子句根据值匹配两个表中的记录`name`。

  该查询使用a `INNER JOIN`来组合表。一个`INNER JOIN`或者从表许可证行当且仅当两个表满足所规定的条件，以显示在结果 `ON`子句。在这个例子中， `ON`子句指定 `name`列中的 `pet`表必须的匹配 `name`列 `event`表。如果名称出现在一个表中但不出现在另一个表中，则该行不会出现在结果中，因为该`ON` 子句中的条件失败。
- 由于`name`列出现在两个表中，因此您必须具体说明引用该列时的表。这是通过将表名添加到列名称来完成的。

您不需要两个不同的表来执行连接。如果要将表中的记录与同一表中的其他记录进行比较，有时将表连接到自身会很有用。例如，要在您的宠物中找到繁殖对，您可以`pet`自己加入表格，以生成类似物种的男性和女性候选对：

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1 INNER JOIN pet AS p2
    ->   ON p1.species = p2.species AND p1.sex = 'f' AND p2.sex = 'm';
+--------+------+--------+------+---------+
| name   | sex  | name   | sex  | species |
+--------+------+--------+------+---------+
| Fluffy | f    | Claws  | m    | cat     |
| Buffy  | f    | Fang   | m    | dog     |
| Buffy  | f    | Bowser | m    | dog     |
+--------+------+--------+------+---------+
```

在此查询中，我们为表名指定别名以引用列，并保持每个列引用与表关联的表的实例。

3.3.4从表中检索信息

该SELECT语句用于从表中提取信息。声明的一般形式是：

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

what_to_select表示你想看到什么。这可以是列列表，也 *可以表示“ 所有列”。“ ***which_table***表示要从中检索数据的表。该WHERE 条款是可选的。如果存在，则 ***conditions_to_satisfy***指定行必须满足的一个或多个条件才有资格进行检索。

3.3创建和使用数据库

一旦知道如何输入SQL语句，就可以访问数据库了。

假设您家中有几只宠物（您的动物园），并且您希望跟踪有关它们的各种类型的信息。您可以通过创建表来保存数据并使用所需信息加载数据。然后，您可以通过从表格中检索数据来回答有关您的动物的各种问题。本节介绍如何执行以下操作：

- 创建一个数据库
- 创建一个表
- 将数据加载到表中
- 以各种方式从表中检索数据
- 使用多个表

动物园数据库很简单（故意），但要想到可能使用类似数据库的真实情况并不难。例如，农民可以使用这样的数据库来跟踪牲畜，或者由兽医跟踪患者记录。可以从MySQL网站获得包含以下部分中使用的一些查询和样本数据的动物园分布。它可以通过https://dev.mysql.com/doc/以压缩**tar**文件和Zip格式提供。

使用该SHOW语句查找服务器上当前存在的数据库：

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

该mysql数据库描述了用户访问权限。该test数据库通常可作为用户工作区尝试的事情了。

声明显示的数据库列表可能与您的计算机不同; SHOW DATABASES 如果您没有该SHOW DATABASES 权限，则不会显示您没有权限的数据库。请参见第13.7.6.14节“显示数据库语法”。

如果test数据库存在，请尝试访问它：

```
mysql> USE test
Database changed
```

USE，比如QUIT，不需要分号。（如果你愿意，可以用分号终止这些语句;它没有任何害处。）这种 USE说法在另一种方式上也是特殊的：它必须在一行上给出。

您可以使用test数据库（如果您有权访问它）来获取后面的示例，但是您在该数据库中创建的任何内容都可以被有权访问它的任何其他人删除。因此，您可能应该要求您的MySQL管理员允许使用您自己的数据库。假设你想打电话给你menagerie。管理员需要执行如下语句：

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

your_mysql_name分配给您的MySQL用户名 在哪里，是your_client_host您连接到服务器的主机。

3.4获取有关数据库和表的信息

如果您忘记了数据库或表的名称，或者给定表的结构是什么（例如，它的列被调用），该怎么办？MySQL通过几个语句来解决这个问题，这些语句提供有关它支持的数据库和表的信息。

您之前看到过`SHOW DATABASES`，它列出了服务器管理的数据库。要找出当前选择的数据库，请使用以下`DATABASE()`函数：

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

如果您尚未选择任何数据库，则结果为`NULL`。

要找出默认数据库包含的表（例如，当您不确定表的名称时），请使用以下语句：

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_menagerie |
+---------------------+
| event               |
| pet                 |
+---------------------+
```

此语句生成的输出中的列名称始终为 ，其中是数据库的名称。有关更多信息，请参见第13.7.6.37节“SHOW TABLES语法”。 `Tables_in_`***`db_name`*** ***`db_name`***

如果你想了解一个表的结构，那么该`DESCRIBE`语句是有用的; 它显示有关每个表列的信息：

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

`Field`表示列名称，`Type`是列 的数据类型，`NULL`指示列 是否可以包含 `NULL`值，`Key`指示列是否已编制索引，并`Default` 指定列的默认值。`Extra` 显示有关列的特殊信息：如果使用该`AUTO_INCREMENT`选项创建列，则值将为`auto_increment`空而不是空。

`DESC`是一种简短的形式 `DESCRIBE`。有关更多信息，请参见 第13.8.1节“DESCRIBE语法”。

您可以`CREATE TABLE` 使用该`SHOW CREATE TABLE`语句获取创建现有表所需的 语句。请参见 第13.7.6.10节“SHOW CREATE TABLE语法”。

如果表上有索引，则生成有关它们的信息。有关此语句的更多信息，请参见第13.7.6.22节“SHOW INDEX语法”。 `SHOW INDEX FROM` ***`tbl_name`***

3.5在批处理模式下使用mysql

在前面的部分中，您以 交互方式使用**mysql**输入语句并查看结果。您也可以在批处理模式下运行**mysql**。为此，将要运行的语句放在文件中，然后告诉 **mysql**从文件中读取其输入：

```
shell> mysql < batch-file
```

如果您在Windows下运行**mysql**并且文件中有一些特殊字符会导致问题，您可以这样做：

```
C:\> mysql -e "source batch-file"
```

如果需要在命令行上指定连接参数，则命令可能如下所示：

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

当您以这种方式使用**mysql时**，您将创建一个脚本文件，然后执行该脚本。

如果您希望脚本继续运行，即使其中的某些语句产生错误，您也应该使用 `--force`命令行选项。

为什么要使用脚本？原因如下：

- 如果重复运行查询（例如，每天或每周），使其成为脚本使您可以避免每次执行时重新输入。
- 您可以通过复制和编辑脚本文件从现有的查询中生成新查询。
- 在开发查询时，批处理模式也很有用，特别是对于多行语句或多语句序列。如果你犯了一个错误，你不必重新输入所有内容。只需编辑脚本以更正错误，然后告诉**mysql**再次执行它。
- 如果您有一个产生大量输出的查询，您可以通过寻呼机运行输出，而不是看着它从屏幕顶部滚动：

  ```
  shell> mysql < batch-file | more
  ```

- 您可以捕获文件中的输出以进行进一步处理：

  ```
  shell> mysql < batch-file > mysql.out
  ```

- 您可以将脚本分发给其他人，以便他们也可以运行语句。

- 某些情况不允许交互式使用，例如，当您从**cron**作业运行查询时。在这种情况下，您必须使用批处理模式。

在批处理模式下 运行**mysql**时，默认输出格式与交互使用时的默认输出格式不同（更简洁）。例如，`SELECT DISTINCT species FROM pet`当**mysql**以交互方式运行时，输出看起来像这样 ：

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

在批处理模式下，输出看起来像这样：

```
species
bird
cat
dog
hamster
snake
```

如果要以批处理模式获取交互式输出格式，请使用**mysql -t**。要回显输出执行的语句，请使用**mysql -v**。

您还可以使用命令或 命令从**mysql**提示符中使用脚本： `source\.`

```
mysql> source filename;
mysql> \. filename
```

有关更多信息，请参见第4.5.1.5节“从文本文件执行SQL语句”。

3.6.1列的最大值

“什么是最高的项目编号？”

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2保持某列最大值的行

任务：查找最昂贵文章的编号，经销商和价格。

这可以通过子查询轻松完成：

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

其他解决方案是使用a `LEFT JOIN`或者按价格降序排序所有行，并使用特定于MySQL的`LIMIT`子句获取第一行：

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

> **注意**
>
> 如果有几个最昂贵的文章，每个价格为19.95，`LIMIT`解决方案只显示其中一个。

3.6.3每组最大列数

任务：找到每篇文章的最高价格。

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article;

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4保持某一列的分组最大值的行

任务：对于每篇文章，找到价格最贵的经销商或经销商。

这个问题可以通过像这样的子查询来解决：

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

前面的示例使用相关子查询，这可能是低效的（请参见第13.2.11.7节“相关子查询”）。解决问题的其他可能性是在`FROM`子句中使用不相关的子查询或a `LEFT JOIN`。

不相关的子查询：

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
  ON s1.article = s2.article AND s1.price = s2.price;
```

`LEFT JOIN`：

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL;
```

的`LEFT JOIN`基础上的作品，当 `s1.price`是在其最大价值，不存在 `s2.price`具有更大价值，从而相应的`s2.article`值 `NULL`。请参见第13.2.10.2节“JOIN语法”。

3.6.5使用用户定义的变量

您可以使用MySQL用户变量来记住结果，而无需将它们存储在客户端的临时变量中。（参见 第9.4节“用户定义的变量”。）

例如，要查找价格最高和最低的文章，您可以执行以下操作：

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

> **注意**
>
> 也可以在用户变量中存储数据库对象（如表或列）的名称，然后在SQL语句中使用此变量。但是，这需要使用准备好的声明。有关更多信息，请参见 第13.5节“准备的SQL语句语法”。

3.6.6使用外键

在MySQL中，InnoDB表支持检查外键约束。请参见 第15章，*InnoDB存储引擎*和 第1.8.2.3节“外键差异”。

仅仅为了连接两个表，不需要外键约束。对于除以外的存储引擎 InnoDB，可以在定义列时使用 没有实际效果的子句，并且*仅作为备忘录或注释，您当前定义的列旨在引用列中的列。另一张桌子*。在使用以下语法时要意识到： REFERENCES ***tbl_name***(***col_name***)

- MySQL不执行任何类型的操作CHECK 以确保***col_name*** 实际存在***tbl_name***（或者甚至它***tbl_name***本身存在）。
- MySQL不执行任何类型的操作， ***tbl_name***例如删除行以响应对您定义的表中的行所采取的操作; 换句话说，这种语法不会引起任何行为ON DELETE或ON UPDATE行为。（虽然您可以编写ON DELETE或ON UPDATE子句作为子句的一部分REFERENCES，但也会被忽略。）
- 此语法创建一个*列*; 它并 ***不会***创建任何类型的索引或关键的。

您可以使用如此创建的列作为连接列，如下所示：

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
```

```
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+


SELECT s.* FROM person p INNER JOIN shirt s
   ON s.owner = p.id
 WHERE p.name LIKE 'Lilliana%'
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

当以这种方式使用时，REFERENCES 子句中不显示的输出 SHOW CREATE TABLE或 DESCRIBE：

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,

```

```
10  PRIMARY KEY  (`id`)
    ) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4
```

REFERENCES以这种方式 使用列定义中的注释或“ 提醒 ”可以使用MyISAM表。

3.6.7搜索两个密钥

一个OR使用单个密钥被很好地优化，因为是的处理 AND。

一个棘手的案例是搜索两个不同的键并结合OR：

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

这种情况已经过优化。请参见 第8.2.1.3节“索引合并优化”。

您还可以使用UNION结合两个单独SELECT语句的输出的有效方法来解决问题 。请参见第13.2.10.3节“UNION语法”。

每个SELECT只搜索一个键，可以优化：

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

3.6.8计算每日访问量

以下示例显示如何使用位组功能计算用户访问网页的每月天数。

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,
             day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
            (2000,2,23),(2000,2,23);
```

示例表包含表示用户对页面的访问的年 - 月 - 日值。要确定这些访问每月发生的天数，请使用以下查询：

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

哪个回报：

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

该查询计算每个年/月组合在表中显示的天数，并自动删除重复的条目。

3.6.9使用AUTO_INCREMENT

该AUTO_INCREMENT属性可用于为新行生成唯一标识：

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

哪个回报：

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

没有为AUTO_INCREMENT 列指定值，因此MySQL会自动分配序列号。除非NO_AUTO_VALUE_ON_ZERO启用了SQL模式，否则您还可以向列显式指定0以生成序列号 。例如：

```
INSERT INTO animals (id,name) VALUES(0,'groundhog');
```

如果声明了列NOT NULL，则还可以分配NULL给列以生成序列号。例如：

```
INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
```

当您将任何其他值插入 AUTO_INCREMENT列时，该列将设置为该值并重置序列，以便下一个自动生成的值按顺序从最大列值开始。例如：

```
INSERT INTO animals (id,name) VALUES(100,'rabbit');
INSERT INTO animals (id,name) VALUES(NULL,'mouse');
SELECT * FROM animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
|   3 | penguin   |
|   4 | lax       |
|   5 | whale     |
|   6 | ostrich   |
|   7 | groundhog |
|   8 | squirrel  |
| 100 | rabbit    |
| 101 | mouse     |
+-----+-----------+
```

更新现有AUTO_INCREMENT列值也会重置AUTO_INCREMENT 序列。

您可以AUTO_INCREMENT使用LAST_INSERT_ID()SQL函数或mysql_insert_id()C API函数检索最新自动生成的 值 。这些函数是特定于连接的，因此它们的返回值不受另一个同时执行插入的连接的影响。

对于AUTO_INCREMENT足够大的列，请使用最小的整数数据类型， 以保存所需的最大序列值。当列达到数据类型的上限时，下一次生成序列号的尝试将失败。UNSIGNED如果可能，请使用该 属性以允许更大的范围。例如，如果你使用 TINYINT，允许的最大序列号为127的 TINYINT UNSIGNED，最大值为255见 第11.2.1节“整型（精确值） - INTEGER，INT，SMALLINT，TINYINT，MEDIUMINT，BIGINT”为所有整数类型的范围。

> **注意**
>
> 对于多行插入， LAST_INSERT_ID()而 mysql_insert_id()实际返回AUTO_INCREMENT从钥匙 *第*一插入行的。这使得多行插入可以在复制设置中的其他服务器上正确再现。

要以AUTO_INCREMENT1以外的值开头，请使用CREATE TABLE或设置该值ALTER TABLE，如下所示：

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

InnoDB备注

有关AUTO_INCREMENT特定用途的信息InnoDB，请参见 第15.8.1.5节“InnoDB中的AUTO_INCREMENT处理”。

MyISAM笔记

- 对于MyISAM表，您可以AUTO_INCREMENT在多列索引中的辅助列上指定 。在这种情况下，AUTO_INCREMENT列的生成值计算为 。当您想要将数据放入有序组时，这非常有用。MAX(***auto_increment_column***) + 1 WHERE prefix=***given-prefix***

```
CREATE TABLE animals (
    grp ENUM('fish','mammal','bird') NOT NULL,
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (grp,id)
) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES
    ('mammal','dog'),('mammal','cat'),
    ('bird','penguin'),('fish','lax'),('mammal','whale'),
    ('bird','ostrich');

SELECT * FROM animals ORDER BY grp,id;
```

  哪个回报：

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

  在这种情况下（当AUTO_INCREMENT 列是多列索引的一部分时），AUTO_INCREMENT如果删除AUTO_INCREMENT任何组中具有最大值的行，则重用 值。即使对于通常不会重用MyISAM其AUTO_INCREMENT值的表也会发生这种情况。

- 如果AUTO_INCREMENT列是多个索引的一部分，则MySQL使用以AUTO_INCREMENT列开头的索引（如果有）生成序列值 。例如，如果该animals表包含索引PRIMARY KEY (grp, id)和INDEX (id)时，MySQL将忽略 PRIMARY KEY用于产生序列的值。结果，该表将包含单个序列，而不是每个grp值的序列。

进一步阅读

有关更多信息，AUTO_INCREMENT请访问：

- 如何将AUTO_INCREMENT 属性分配给列：第13.1.18节“CREATE TABLE语法”和 第13.1.8节“ALTER TABLE语法”。
- 如何AUTO_INCREMENT取决于行为会 NO_AUTO_VALUE_ON_ZERO SQL模式：第5.1.10，“SQL服务器模式”。
- 如何使用该 LAST_INSERT_ID()函数查找包含最新AUTO_INCREMENT值的行 ： 第12.14节“信息函数”。
- 设置AUTO_INCREMENT要使用的值：第5.1.7节“服务器系统变量”。
- 第15.8.1.5节“InnoDB中的AUTO_INCREMENT处理”
- AUTO_INCREMENT和复制： 第17.4.1.1节“复制和AUTO_INCREMENT”。
- 与AUTO_INCREMENT（ auto_increment_increment 和 auto_increment_offset ）相关的服务器系统变量 ，可用于复制： 第5.1.7节“服务器系统变量”。

3.6常见查询示例

以下是如何解决MySQL的一些常见问题的示例。

一些示例使用该表`shop`来保存某些交易商（交易商）的每个商品（商品编号）的价格。假设每个交易者每篇文章只有一个固定价格，则（`article`，`dealer`）是记录的主键。

启动命令行工具**mysql**并选择一个数据库：

```
shell> mysql your-database-name
```

（在大多数MySQL安装中，您可以使用名为的数据库`test`）。

您可以使用以下语句创建和填充示例表：

```
CREATE TABLE shop (
    article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
    dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
    price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
    PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
    (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

发出语句后，该表应具有以下内容：

```
SELECT * FROM shop;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | A      |  3.45 |
```

```
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | B      |  1.45 |
|    0003 | C      |  1.69 |
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

3.7在Apache中使用MySQL

有些程序可以让您从MySQL数据库中验证用户身份，还可以将日志文件写入MySQL表。

您可以通过将以下内容添加到Apache配置文件中来更改Apache日志记录格式，以便MySQL可以轻松读取：

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

要将该格式的日志文件加载到MySQL中，您可以使用如下语句：

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

应创建指定的表，以使列与`LogFormat`行写入日志文件的列相对应。

第4章MySQL程序

目录

本章简要概述了Oracle Corporation提供的MySQL命令行程序。它还讨论了运行这些程序时指定选项的一般语法。大多数程序具有特定于其自身操作的选项，但所有选项的语法类似。最后，本章提供了有关各个程序的更详细说明，包括它们识别的选项。