

目錄

教程目录	1.1
第一章 基础教程	1.2
第一节 入门	1.2.1
1.1 安装	1.2.1.1
1.2 示例数据库	1.2.1.2
第二节 查询数据	1.2.2
2.1 SELECT 语句	1.2.2.1
2.2 SELECT DISTINCT 语句	1.2.2.2
第三节 过滤数据	1.2.3
3.1 WHERE 子句	1.2.3.1
3.2 AND 运算符	1.2.3.2
3.3 OR 运算符	1.2.3.3
3.4 IN 运算符	1.2.3.4
3.5 BETWEEN 运算符	1.2.3.5
3.6 LIKE 运算符	1.2.3.6
3.7 LIMIT 子句	1.2.3.7
3.8 IS NULL 运算符	1.2.3.8
第四节 数据排序	1.2.4
4.1 ORDER BY 子句	1.2.4.1
4.1 自然排序	1.2.4.2
第五节 连接表	1.2.5
5.1 别名	1.2.5.1
5.2 INNER JOIN	1.2.5.2
5.3 LEFT JOIN	1.2.5.3
5.4 自连接	1.2.5.4
5.5 CROSS JOIN	1.2.5.5
第六节 分组数据	1.2.6

6.1 GROUP BY子句	1.2.6.1
6.2 HAVING 子句	1.2.6.2
第七节 子查询，派生表和通用表达式	1.2.7
7.1 子查询	1.2.7.1
7.2 派生表	1.2.7.2
7.3 公共表表达式	1.2.7.3
7.4 递归 CTE	1.2.7.4
第八节 集合操作符	1.2.8
8.1 UNION 和 UNION ALL	1.2.8.1
8.2 INTERSECT 模拟	1.2.8.2
第九节 修改数据	1.2.9
9.1 INSERT 语句	1.2.9.1
9.2 INSERT IGNORE	1.2.9.2
9.3 UPDATE 语句	1.2.9.3
9.4 UPDATE JOIN 语句	1.2.9.4
9.5 DELETE	1.2.9.5
9.6 ON DELETE CASCADE.	1.2.9.6
9.7 DELETE JOIN	1.2.9.7
9.8 REPLACE 语句	1.2.9.8
9.9 PREPARE 语句	1.2.9.9
第十节 事务	1.2.10
10.1 事务介绍	1.2.10.1
10.2 表锁定	1.2.10.2
第十一节 管理数据库和表	1.2.11
11.1 数据库管理	1.2.11.1
11.2 MySQL 表类型	1.2.11.2
11.3 CREATE TABLE	1.2.11.3
11.4 序列	1.2.11.4
11.5 ALTER TABLE	1.2.11.5
11.6 重命名表	1.2.11.6

11.7 从表中删除列	1.2.11.7
11.8 向表中添加新列	1.2.11.8
11.9 删除表	1.2.11.9
11.10 临时表	1.2.11.10
11.11 TRUNCATE TABLE	1.2.11.11
第十二节 索引	1.2.12
12.1 管理索引	1.2.12.1
12.1 UNIQUE 索引	1.2.12.2
第十三节 数据类型	1.2.13
13.1 INT	1.2.13.1
13.2 DECIMAL	1.2.13.2
13.3 BIT	1.2.13.3
13.4 BOOLEAN	1.2.13.4
13.5 CHAR	1.2.13.5
13.6 VARCHAR	1.2.13.6
13.7 TEXT	1.2.13.7
13.8 DATE	1.2.13.8
13.9 TIME	1.2.13.9
13.10 DATETIME	1.2.13.10
13.11 TIMESTAMP	1.2.13.11
13.12 JSON	1.2.13.12
13.13 ENUM	1.2.13.13
第十四节 约束	1.2.14
14.1 NOT NULL 约束	1.2.14.1
14.2 主键约束	1.2.14.2
14.2 外键约束	1.2.14.3
14.4 UNIQUE 约束	1.2.14.4
14.5 CHECK 约束	1.2.14.5
第十五节 全球化	1.2.15
15.1 字符集	1.2.15.1

15.2 校对规则	1.2.15.2
第十六节 导入和导出	1.2.16
16.1 导入 CSV 文件	1.2.16.1
16.2 导出为 CSV	1.2.16.2
第二章 技巧	1.3
CTE 简介	1.3.1
递归 CTE	1.3.2
邻接列表模型和层次结构	1.3.3
获取行数	1.3.4
比较表	1.3.5
找重复值	1.3.6
删除重复行	1.3.7
UUID 和主键	1.3.8
复制表数据	1.3.9
变量	1.3.10
生成列	1.3.11
连续行比较	1.3.12
更改存储引擎	1.3.13
基于正则表达式的搜索	1.3.14
row_number 模拟	1.3.15
随机选择记录	1.3.16
选择第 n 个最高纪录	1.3.17
重置自动增量值	1.3.18
MariaDB 与 MySQL 比较	1.3.19
间隔	1.3.20
NULL 详细和应用	1.3.21
获取今天的日期	1.3.22
将NULL值映射到有意义的值	1.3.23
注释	1.3.24
第三章 存储过程	1.4

简介	1.4.1
入门	1.4.2
变量	1.4.3
参数	1.4.4
返回多个值	1.4.5
IF语句	1.4.6
CASE语句	1.4.7
IF和CASE语句的技巧	1.4.8
循环	1.4.9
游标	1.4.10
列出存储过程	1.4.11
错误处理	1.4.12
SIGNAL 和 ESIGNAL 语句	1.4.13
存储函数	1.4.14
第四章 触发器	1.5
实现	1.5.1
创建	1.5.2
创建多个触发器	1.5.3
管理	1.5.4
计划事件	1.5.5
修改事件	1.5.6
第五章 视图	1.6
简介	1.6.1
实现和限制	1.6.2
创建	1.6.3
可更新视图	1.6.4
确保视图一致性	1.6.5
检查选项子句	1.6.6
管理	1.6.7
第六章 全文搜索	1.7

简介	1.7.1
定义索引	1.7.2
自然语言全文搜索	1.7.3
布尔全文搜索	1.7.4
查询扩展	1.7.5
ngram全文解析器	1.7.6
第七章 函数	1.8
第八章 管理	1.9
访问控制系统入门	1.9.1
创建用户	1.9.2
授予权限	1.9.3
撤销权限	1.9.4
角色管理	1.9.5
删除用户	1.9.6
维护数据库表	1.9.7
备份数据库	1.9.8
列出数据库	1.9.9
列出表	1.9.10
列出表的列	1.9.11
列出用户	1.9.12

备注：非本人原创，本人在学习 MySQL 过程中发现这个网站 <http://www.yiibai.com> 的 MySQL 教程挺好的，只是排版混乱，无法快速查阅，于是重新排版，方便查看。

在线阅读，请点击：<https://legacy.gitbook.com/book/necan/mysql-tutorial/details>

教程目录

开发人员的 MySQL 教程

如果你是一个寻找学习 MySQL 的 web 开发人员，在本节中，您可立即开始学习使用 MySQL，并学习如何有效地使用 MySQL 来更有效地完成您的工作。

如果您浏览整个教程，了解如何使用如 SQL 查询，MySQL 存储过程，数据库视图，触发器等各种技术来管理 MySQL 数据库和操作数据。

第一章 基础教程

- 本节将帮助您熟悉基本的 MySQL 知识，包括使用各种 SQL 语句（如 `INSERT`，`DELETE`，`UPDATE` 和 `SELECT`）来管理 MySQL 数据库和操作数据。还将了解高级数据查询技术，包括 `INNER JOIN`，`LEFT JOIN`，子查询，`UNION` 等。参考阅读：<http://www.yiibai.com/mysql/basic-mysql.html>

第二章 常用技巧

- 本节将提供一些高级的 MySQL 技术和技巧，以帮助您有效解决 MySQL 中遇到的一些棘手的问题。参考阅读：<http://www.yiibai.com/mysql/mysqltips.html>

第三章 存储过程

- 在本节中，您将学习如何在 MySQL 中创建存储过程，并通过明确的说明和示例演示如何使用存储过程。参考阅读：<http://www.yiibai.com/mysql/stored-procedure.html>

第四章 触发器

- MySQL 触发器是自动执行以响应与表相关联的特定事件的存储程序，例如插入，更新或删除记录。本节介绍如何使用 MySQL 数据库触发器。参考阅读：<http://www.yiibai.com/mysql/triggers.html>

第五章 视图

- 在本节中，您将了解数据库视图，如何创建数据库视图并在 MySQL 中进行管理。参考阅读：<http://www.yiibai.com/mysql/views.html>

第六章 全文搜索

- 在本节中，演示如何使用 MySQL 全文搜索与各种全文搜索技术，如自然语言搜索，布尔语言搜索和查询扩展。参考阅读：<http://www.yiibai.com/mysql/full-text-search.html>

第七章 MySQL 函数

本节提供最常用的 MySQL 函数，包括聚合函数，字符串函数，日期和时间函数以及控制流函数使用和示例。参考阅读：<http://www.yiibai.com/mysql/functions.html>

数据库管理员的 MySQL 教程

这个分步教程为您提供了有关 MySQL 管理更深层次的信息。这里涵盖了从基础到高级 MySQL 管理和配置的一切知识。本节中介绍的所有 MySQL 管理教程都是很实用的，您可在企业生产环境中应用(使用)。

第八章 MySQL 管理

在本节中，您将找到许多有用的 MySQL 管理教程，包括 MySQL 服务器启动和关闭，MySQL 服务器安全性，MySQL 数据库维护，备份和复制。参考阅读：<http://www.yiibai.com/mysql/administration.html>

第一章 基础教程

这个 MySQL 基础教程解释一些基本的 SQL 语句。如果这是您第一次使用关系数据库管理系统，本教程将为您提供使用 MySQL 数据库服务器所需的一切内容，例如查询数据，更新数据，管理数据库和创建表。

如果您已经熟悉其他关系数据库管理系统(如 PostgreSQL，Oracle 或 Microsoft SQL Server 等)，则可以使用本教程来刷新您的知识，并了解 MySQL 的 SQL 方言与其他数据库系统的不同之处。

第 1 节 入门

本节将开始介绍和学习使用 MySQL。我们将开始安装 MySQL，下载[示例数据库](#)并将数据导入到 MySQL 服务器以进行练习。

- 安装MySQL数据库服务器 - 演示如何在计算机上安装 MySQL 数据库服务器。
- 下载MySQL示例数据库 - 介绍一个名称为 yiibaidb 的 MySQL 示例数据库，提供下载示例数据库及 ER 图。
- 将示例数据库导入到MySQL数据库服务器中 - 演示如何将示例数据库 (yiibaidb) 导入到 MySQL 数据库服务器中进行练习。

第 2 节 查询数据

本节将帮助您了解如何从 MySQL 数据库服务器查询数据。我们将从一个简单的 SELECT 语句开始，从单个表查询数据。

- SELECT 语句 - 显示如何使用简单的 SELECT 语句来查询单个表中的数据。
- SELECT DISTINCT 语句 - 了解如何在 SELECT 语句中使用 DISTINCT 运算符来消除结果集中的重复行。

第 3 节 过滤数据

- WHERE - 学习如何使用 WHERE 子句根据指定的条件过滤行记录。
- AND 运算符 - 介绍如何使用 AND 运算符以组合布尔表达式以形成用于过滤数据的复杂条件。

- **OR 运算符** - 介绍 `OR` 运算符，并展示如何将 `OR` 运算符与 `AND` 运算符组合以过滤数据。
- **IN 运算符** - 学习如何在 `WHERE` 子句中使用 `IN` 运算符来确定值是否匹配列表或子查询中的指定值。
- **BETWEEN 运算符** - 显示如何使用 `BETWEEN` 运算符来根据指定范围查询数据。
- **LIKE** - 提供基于特定模式匹配查询数据的技术示例，以执行一些模糊查询。
- **LIMIT 子句** - 使用 `LIMIT` 来限制 `SELECT` 语句返回的行数
- **IS NULL** - 使用 `IS NULL` 运算符测试值是否为 `NULL`。

第 4 节 排序数据

- **ORDER BY** - 显示如何使用 `ORDER BY` 子句排序结果集。还将介绍使用 `FIELD` 函数的自定义排序顺序。
- **使用 ORDER BY 子句进行自然排序** - 通过使用 `ORDER BY` 子句，演示 MySQL 中的各种自然排序技术。

第 5 节 连接表

- **MySQL 别名** - 引入别名，包括表别名和列别名，以提高复杂查询的可读性，并避免在查询具有相同列名称的多个表中的数据时发生歧义错误。
- **INNER JOIN** - 应用内部连接技术来查询来自多个相关表的数据。
- **LEFT JOIN** - 学习如何使用左连接来生成包含来自连接左侧表中的行的结果集，并使用 `NULL` 值来补充不匹配行。
- **CROSS JOIN** - 学习如何使来自多个表的行的笛卡尔乘积。
- **自连接** - 使用表别名将表连接到自身，并使用其他类型的连接(如 `INNER JOIN` 或 `LEFT JOIN`)连接同一表中的行记录。

第 6 节 分组数据

- **GROUP BY 子句** - 学习如何根据列或表达式将行记录分组到子组。
- **HAVING 子句** - 按特定条件过滤组。

第 7 节 MySQL 子查询，派生表和通用表达式

- **MySQL 子查询** - 学习如何在另一个查询(外部查询)中嵌套另一个查询语句(内部查询)，并使用内部查询的结果值作为外部查询条件。
- **MySQL 派生表** - 介绍派生表概念，并演示如何使用它来简化复杂查询。
- **MySQL 通用表表达式** - 解释通用表表达式概念，并向您展示如何使用 CTE 查询表中的数据。
- **递归 CTE** - 演示如何使用递归通用表表达式 (CTE) 遍历分层数据。

第 8 节 使用 SET 操作符

- **UNION 和 UNION ALL** - 使用 UNION 和 UNION ALL 操作符将两个或多个多个 SELECT 语句的结果集合合并到一个结果集中。
- **INTERSECT 模拟** - 显示了几种模拟MySQL中 INTERSECT 运算符的方法。

第 9 节 修改 MySQL 中的数据

在本节中，将学习如何使用各种MySQL语句来在表上执行插入，更新和删除数据操作。

- **INSERT 语句** - 学习如何使用各种形式的 INSERT 语句将数据插入到数据库表中。
- **INSERT IGNORE** - 解释将数据行插入到表中并忽略导致错误或异常的行的 INSERT IGNORE 语句。
- **UPDATE 语句** - 了解如何使用 UPDATE 语句及其选项来更新数据库表中的数据。
- **UPDATE JOIN 语句** - 显示如何使用带有 INNER JOIN 和 LEFT JOIN 的 UPDATE JOIN 语句执行交叉表更新。
- **DELETE** - 学习如何使用 DELETE 语句从一个或多个表中删除数据。
- **ON DELETE CASCADE** - 学习如何从父表中删除数据时，使用外部键从 DELETE CASCADE 引用动作删除子表中的数据。
- **DELETE JOIN** - 学习如何从多个表中删除数据。
- **REPLACE 语句** - 学习如何插入或更新数据，这取决于数据是否存在于表中。
- **PREPARE 语句** - 显示如何使用PREPARE语句执行查询。

第 10 节 MySQL 事务

- **MySQL事务** - 了解MySQL事务，以及如何使用 COMMIT 和 ROLLBACK 来管理 MySQL中的事务。
- **MySQL表锁定** - 了解如何使用MySQL锁来协调会话之间的表访问。

第 11 节 管理 MySQL 数据库和表

本节介绍如何管理 MySQL 中最重要的数据库对象，包括数据库和表。

- **MySQL 数据库管理** - 学习各种语句来管理 MySQL 数据库，包括创建新数据库，删除现有数据库，选择数据库以及列出所有数据库。
- **MySQL 表类型** - 了解每个表类型的功能至关重要，以便您可以有效地使用它们来最大限度地提高数据库的性能。
- **CREATE TABLE** - 学习如何使用 CREATE TABLE 语句在数据库中创建新表。
- **MySQL 序列** - 学习如何使用序列为表的主键列自动生成唯一的数字。
- **ALTER TABLE** - 学习如何使用 ALTER TABLE 语句来更改现有表的结构。
- **重命名表** - 演示如何使用 RENAME TABLE 语句重命名表。
- **从表中删除列** - 学习如何使用 ALTER TABLE DROP COLUMN 语句从表中删除一个或多个列。
- **向表中添加新列** - 学习如何使用 ALTER TABLE ADD COLUMN 语句向现有表添加一个或多个列。
- **删除表** - 学习如何使用 DROP TABLE 语句删除现有表。
- **MySQL 临时表** - 讨论MySQL临时表，并学习如何管理临时表。
- **TRUNCATE TABLE** - 学习如何使用 TRUNCATE TABLE 语句删除表中的所有数据。

第 12 节 MySQL 索引

- **管理MySQL 数据库索引** - 学习如何使用 MySQL 索引，以及如何利用索引来加快数据检索。
- **MySQL UNIQUE 索引** - 显示如何使用 UNIQUE 索引来强制一个或多个列的值的唯一性。

第 13 节 MySQL 数据类型

- **MySQL 数据类型** - 学习 MySQL 中的各种数据类型，以便您可以在设计数据库

表时有效应用它们。

- **INT** - 学习如何使用整数数据类型。并演示如何使用 `ZEROFILL` 和整数列的宽度属性。
- **DECIMAL** - 学习如何使用 `DECIMAL` 数据类型存储十进制格式的精确值。
- **BIT** - 介绍 `BIT` 数据类型以及如何在 MySQL 中存储位值。
- **BOOLEAN** - 学习 MySQL 如何通过内部使用 `TINYINT(1)` 来处理布尔值。
- **CHAR** - 学习如何使用存储固定长度字符串的 `CHAR` 数据类型。
- **VARCHAR** - 提供 `VARCHAR` 数据类型的基本指南。
- **TEXT** - 演示如何使用 `TEXT` 数据类型存储文本数据。
- **DATE** - 介绍 `DATE` 数据类型，并显示一些日期功能来有效处理日期数据。
- **TIME** - 学习 `TIME` 数据类型的功能，并向您演示如何使用一些有用的时间功能来处理时间数据。
- **DATETIME** - 介绍 `DATETIME` 数据类型和一些有用的函数来操作日期时间值。
- **TIMESTAMP** - 介绍 `TIMESTAMP` 类型及其功能，调用自动初始化和自动更新，允许您为表定义自动初始化和自动更新的列。
- **JSON 格式类型** - 显示如何使用 `JSON` 数据类型来存储 `JSON` 文档。
- **ENUM** - 了解如何正确使用 `ENUM` 数据类型来存储枚举值。

第 14 节 MySQL 约束

- **NOT NULL 约束** - 引入 `NOT NULL` 约束，并显示如何为列定义 `NOT NULL` 约束或将 `NOT NULL` 约束添加到现有列。
- **主键约束** - 指导如何使用主键约束来创建表的主键。
- **外键约束** - 学习外键概念，并逐步显示如何创建和删除外键。
- **UNIQUE 约束** - 显示如何使用 `UNIQUE` 约束来强制表中列或一组列的值的唯一性。
- **CHECK 约束** - 通过各种方式来模拟 MySQL 中的 `CHECK` 约束。

第 15 节 MySQL 全球化

- **MySQL 字符集** - 本教程讨论 MySQL 字符集，并演示如何对字符集执行各种操作。
- **MySQL 排序规则** - 本教程讨论了 MySQL 排序规则，并向您展示了如何为 MySQL 服务器，数据库，表和列设置字符集和排序规则。

第 16 节 MySQL 导入和导出

- 将 CSV 文件导入 MySQL 表 - 演示如何使用 `LOAD DATA INFILE` 语句将 CSV 文件导入 MySQL 表。
- MySQL 导出表到 CSV - 学习如何将 MySQL 表导出为 CSV 文件格式的各种技术。

在学习 MySQL 之前，首先需要了解数据库和 SQL。如果您已经知道数据库和 SQL，那么可以直接跳转到下一章节的学习。

数据库简介

当您想收听最喜欢的歌曲时，可以从智能手机上打开播放列表。在这种情况下，播放列表是数据库就是从数据库中读取出来的。

当您拍摄照片并将其上传到微博，朋友圈等，这样的社交网络中的帐户时，您的照片库就有可能存储在一个数据库中。

当您浏览电子商务网站购买鞋子，衣服等时，您使用购物车就是数据库应用。

数据库无处不在。那么什么是数据库？根据定义，数据库只是一个结构化的数据集合。

数据本质上相互关联，例如，产品属于产品类别并与多个标签相关联。这就是为什么要使用关系数据库。

在关系数据库中，我们使用表对产品，类别，标签等数据进行建模。表包含列和行。它就像一个电子表格(Excel)。

表可以涉及的使用有：一对一，一对多，多对一关系等关系。

因为我们要处理大量的数据，所以需要一种方法来定义数据库，表等，并更有效地处理数据。另外，我们可以将数据转换成数据信息。

所以就需要SQL来处理了。

SQL - 数据库的语言

SQL 代表结构化查询语言 (**Structured Query Language**)。SQL 是用于访问数据库的标准化语言。

ANSI/SQL 定义了SQL 标准。当前版本的 SQL 是 SQL: 2003。每当我们引用 SQL 标准时，指的就是当前的SQL 版本。

SQL 包含三个部分：

- 数据定义语言包含定义数据库及其对象的语句，例如表，视图，触发器，存储

过程等。

- 数据操作语言包含允许您更新和查询数据的语句。
- 数据控制语言允许授予用户权限访问数据库中特定数据的权限。

现在，您了解数据库和 SQL，现在是时候回答下一个问题了...

MySQL 是什么？

My 是 MySQL 的联合创始人 *Monty Widenius* 的女儿的名字。MySQL 是 My 和 SQL 的组合，这就是 MySQL 命名的由来。

MySQL 的官方网址：<http://www.mysql.com/>，MySQL 的社区版本下载地址为：<http://dev.mysql.com/downloads/mysql/>，在写本文时，当前的 MySQL 最新版本是：5.7.18。

MySQL 是一个数据库管理系统，也是一个关系数据库。它是由 Oracle 支持的开源软件。这意味着任何一个人都可以使用 MySQL 而不用支付一毛钱。另外，如果需要，还可以更改其源代码或进行二次开发以满足您的需要。

即使 MySQL 是开源软件，但是可以从 Oracle 购买商业许可证版本，以获得高级支持服务(特殊企业用户需要)。

与其他数据库软件 (如 Oracle 数据库或 Microsoft SQL Server)相比，MySQL 非常容易学习和掌握。

MySQL 可以在各种平台上运行 UNIX，Linux，Windows 等。可以将其安装在服务器甚至桌面系统上。此外，MySQL 是可靠，可扩展和快速的。

如果您开发网站或 Web 应用程序，MySQL 是一个不错的选择(强烈建议使用)。MySQL 是 LAMP 堆栈的重要组成部分，包括 Linux，Apache，MySQL 和 PHP。

本教程将介绍如何使用 MySQL Installer 在 Windows 平台 (Win10) 上安装 MySQL。在学习并按照本教程所示的步骤操作之后，您将有一个 MySQL 数据库服务器在您的系统中，并以此 MySQL 数据库服务器为基础，运行相关工具来学习和实践 MySQL。

下载 MySQL 安装程序

这里想要说的是，安装 MySQL 的方式有好几种，由于文章篇幅的限制，这里只选定一种作为安装演示。具体的安装，可以按照你喜欢的方式来。只要有两种方式：

1. 在线安装版，下载：*mysql-installer-web-community.exe*
2. 离线安装版，下载：*mysql-installer-community.exe*
3. 解压缩版，下载：*Windows (x86, 64-bit), ZIP Archive*

上面的下载安装程序，可以从网址：<http://dev.mysql.com/downloads/mysql/> 找到。

如果要在 Windows 环境中安装 MySQL，使用 MySQL 安装程序是最简单的方法。MySQL 安装程序为您提供了一个易于使用的向导，可帮助您使用以下组件安装 MySQL：

- MySQL 服务器
- 所有可用连接器
- 具有示例数据模型的 MySQL Workbench
- MySQL 通知程序
- Excel 和 Microsoft Visual Studio 的工具
- MySQL 示例数据库
- MySQL 文档

现在，我们一步步来看，如何下载并安装 MySQL 服务器软件。

第一步：下载所需的安装包

打开网址：<http://dev.mysql.com/downloads/mysql/>，如下所示 -

1.1 安装

Generally Available (GA) Releases Development Releases

MySQL Community Server 5.7.18

Select Operating System:
Microsoft Windows

Select OS Version:
All

Looking for previous GA versions?

Recommended Download:

MySQL Installer for Windows
All MySQL Products. For All Windows Platforms. In One Package.



Starting with MySQL 5.6 the MySQL Installer package replaces the standalone MSI packages.

Windows (x86, 32 & 64-bit), MySQL Installer MSI [Go to Download Page >](#)

Other Downloads:

Windows (x86, 32-bit), ZIP Archive (mysql-5.7.18-win32.zip)	5.7.18	303.9M	Download
Windows (x86, 64-bit), ZIP Archive (mysql-5.7.18-winx64.zip)	5.7.18	316.1M	Download

注：因为在编写本教程时，使用的是 Win10 64 位的操作系统，所以这里选择：*Windows (x86, 64-bit), ZIP Archive* 下载

在弹出的第二个页面中，选择点击 “**No thanks, just start my download.**” 跳过注册/登录帐号环节直接下载。如下图所示 -

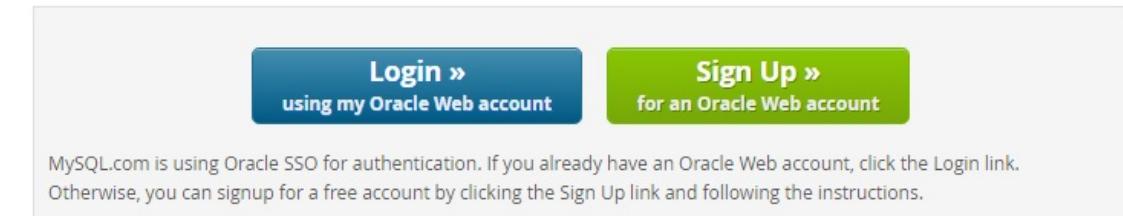
Begin Your Download

mysql-5.7.18-winx64.zip

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system
- Comment in the MySQL Documentation



然后，耐心等待下载完成...

第二步：压缩到指定目录

准备好一个安装 MySQL 程序的目录，如：`D:/software`

在本教程中，您已经学会了如何使用 MySQL 安装程序在 Windows 系统中安装 MySQL，[下载 MySQL 示例数据库](#)并将其加载到 MySQL 服务器中，以通过此 MySQL 教程练习和学习 MySQL。

这里解压后的目录为：`D:\software\mysql-5.7.18-winx64`，此目录下的文件如下所示 -

A screenshot of a Windows File Explorer window. The title bar says 'mysql-5.7.18-winx64'. The address bar shows the path 'D:\software\mysql-5.7.18-winx64'. The file list shows the following files and folders:

名称	修改日期	类型	大小
bin	2017/7/14 22:30	文件夹	
docs	2017/7/14 22:30	文件夹	
include	2017/7/14 22:30	文件夹	
lib	2017/7/14 22:30	文件夹	
share	2017/7/14 22:30	文件夹	
COPYING	2017/3/18 8:45	文件	18 KB
README	2017/3/18 8:45	文件	3 KB

第三步：启动 MySQL 服务器

以管理员身份打开命令行，进入 MySQL 服务器安装的目录：`D:\software\mysql-5.7.18-winx64\bin`，执行以下命令启动 MySQL

```
Microsoft Windows [版本 10.0.10586]
(c) 2015 Microsoft Corporation。保留所有权利。
C:\WINDOWS\system32>D:
D:>cd software\mysql-5.7.18-winx64\bin
D:\software\mysql-5.7.18-winx64\bin>mysqld -install
Service successfully installed.
D:\software\mysql-5.7.18-winx64\bin>net start mysql
MySQL 服务正在启动 ...
MySQL 服务已经启动成功。
D:\software\mysql-5.7.18-winx64\bin>
```

如果有提示如下错误

```
D:\software\mysql-5.7.18-winx64\bin> mysqld.exe
mysqld: Can't change dir to 'D:\software\mysql-5.7.18-winx64\data'
\ ' (Errcode: 2 - No such file or directory)
2017-07-14T18:48:51.023897Z 0          TIMESTAMP with implicit
DEFAULT value is deprecated. Please use --explicit_defaults_for_
timestamp server option (see documentation for more details).

2017-07-14T18:48:51.039516Z 0          Binlog end
2017-07-14T18:48:51.039516Z 0          mysqld.exe: Shutdown comple
te
```

上面错误提示中，已经说明了：

```
mysqld: Can't change dir to 'D:\software\mysql-5.7.18-winx64\data'
\ ' (Errcode: 2 - No such file or directory)
```

则需要在创建一个目录：`D:\software\mysql-5.7.18-winx64\data\`，现在我们就来创建这个目录，在创建目录完成后重新执行 `mysqld -install` 启动 MySQL 服务器 -

一定要注意两个问题，切记！

第一：以管理员自身份打开 CMD； 第二：用 cd 命令进入到你 MySQL 文件的解压路径。

第四步：连接 MySQL 服务器

服务启动成功之后，就可以连接/登录 MySQL 服务器了，打开命令提交符界面输入 mysql -u root -p 或 mysql -h localhost -u root -p (第一次登录没有密码，直接按回车过), 登录成功，但是登录成功后，不能执行任何操作，MySQL 服务器要求您必须设置密码再执行其它操作。

假设我们登录成功后，要查看当前目录下所数据名称(执行查询： show databases)，但它提示要先设置密码。完整的过程如下所示 -

```
D:\software\mysql-5.7.18-winx64\bin>mysql -hlocalhost -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.9

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
ERROR 1820 (HY000): You must reset your password using ALTER USER
R statement before executing this statement.
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.08 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
+-----+
6 rows in set (0.11 sec)
```

有关解压包安装 MySQL 服务器就这样完成了，如果想要使用在线安装的方式来安装 MySQL，请参考：<http://dev.mysql.com/doc/refman/5.7/en/mysql-installer.html>

在本 MySQL 教程中，大部分操作是基于 `yiibaidb` 这个数据库作为学习 MySQL 示例数据库，这样的话有助于您快速有效地使用 MySQL。`yiibaidb` 数据库是一个典型汽车零售商数据库模型。它包含典型的业务数据，如客户，产品，销售订单，销售订单等。

我们在 MySQL 教程中使用此示例数据库来演示从简单查询到复杂存储过程的许多 MySQL 功能。

下载 MySQL 示例数据库

可以在以下链接中下载本 MySQL 教程所使用的示例数据库 (`yiibaidb`)。

示例数据库下载地址：<http://www.yiibai.com/downloads/yiibaidb.zip>

在解压缩上面的文件后，可以将示例数据库导入到 MySQL 数据库服务器中，方法如下：将示例数据库导入到 MySQL 数据库服务器中，并使用以下 SQL 语句进行测试：

```
USE yiibaidb;
SELECT * FROM customers;
```

上面语句首先将当前数据库切换到 `yiibaidb` 数据库下，并从 `customers` 表查询数据。如果您看到返回的客户数据，说明已成功将示例数据库 (`yiibaidb`) 导入 MySQL 数据库服务器了。

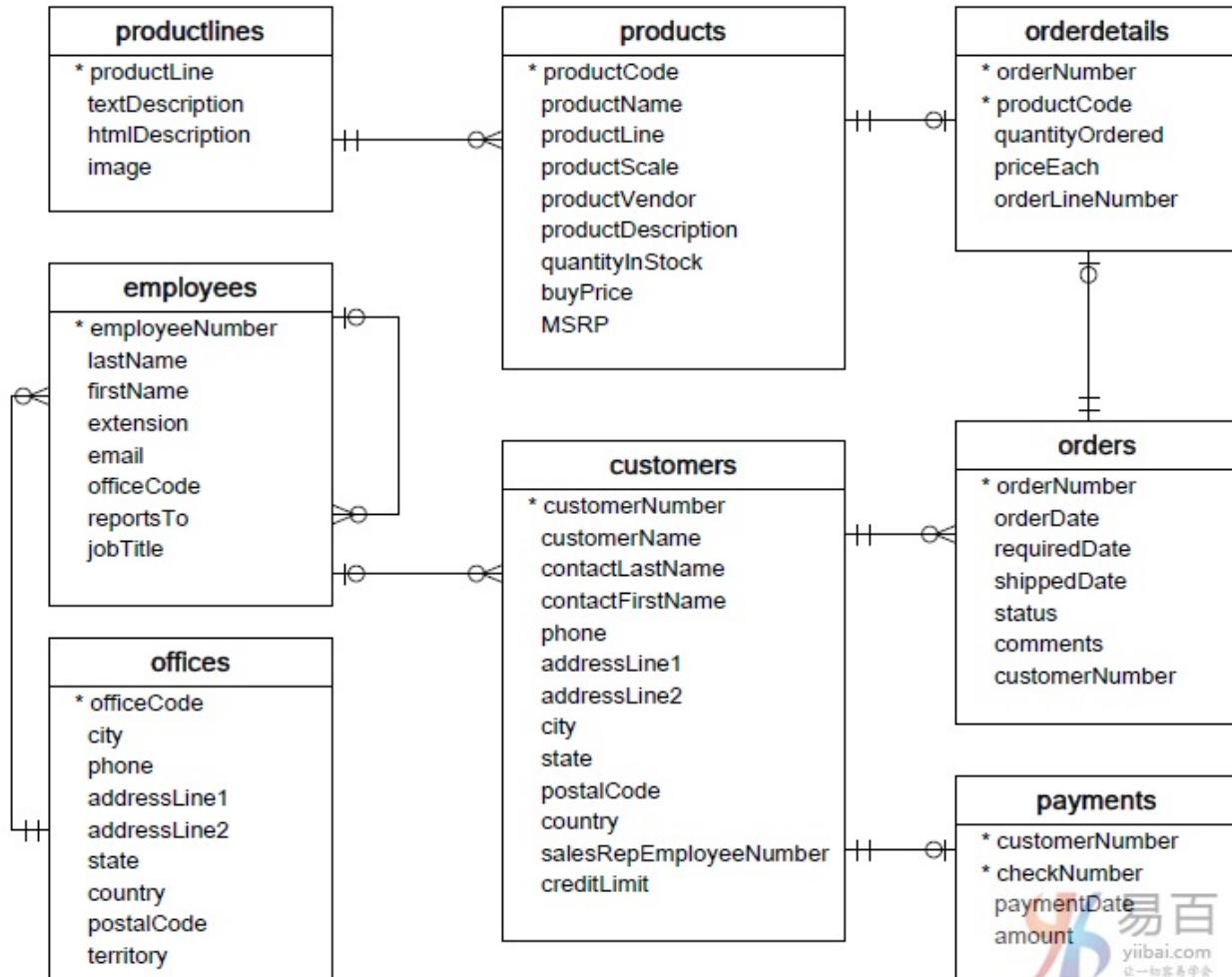
MySQL 示例数据库结构

MySQL 示例数据库模式由以下表组成：

- `customers` : 存储客户的数据。
- `products` : 存储汽车的数据。
- `productLines` : 存储产品类别数据。
- `orders` : 存储客户订购的销售订单。
- `orderDetails` : 存储每个销售订单的订单产品数据项。
- `payments` : 存储客户订单的付款数据信息。
- `employees` : 存储所有员工信息以及组织结构，例如，直接上级(谁向谁报告工作)。

- offices** : 存储销售处数据，类似于各个分公司。

表与表之间的关系，请参考以下ER图 -



我们建议您打印出此 ER 图，并将其粘贴到桌面上，以便在学习 MySQL 的过程中熟悉其中的表之间的关联关系。

导入示例数据库

在本教程中，将学习如何在 MySQL 命令行下将 MySQL 示例数据库导入到 MySQL 数据库服务器中。在本教程之后，您将 **yiibaidb** 示例数据库导入到 MySQL 服务器中以实践和学习 MySQL。

第一步：从 MySQL 示例数据库文章中[下载示例数据库\(yiibaidb \)](#)，有关示例数据库的结构，请参考：<http://www.yiibai.com/mysql/sample-database.html>

第二步：将下载的文件解压缩到临时文件夹中。为了简单起见，我们将把它解压缩到 **D:\worksp**，如下所示 -

1.2 示例数据库

> 此电脑 > Software (D:) > worksp

名称	修改日期	类型	大小
yiibaidb.sql	2017/7/15 3:35	SQL 文件	206 KB



第三步：连接到 MySQL 服务器并创建数据库

```
D:\software\mysql-5.7.18-winx64\bin>mysql -hlocalhost -uroot -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 23
Server version: 5.7.9 MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

创建数据库

```
mysql> CREATE DATABASE IF NOT EXISTS yiibaidb DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
mysql> use yiibaidb;
```

导入数据

```
mysql> use yiibaidb;
mysql> source D:/worksp/yiibaidb.sql;
```

第四步：测试导入结果

```
mysql> select city, phone, country from `offices`;  
+-----+-----+-----+  
| city | phone | country |  
+-----+-----+-----+  
| San Francisco | +1 650 219 4782 | USA |  
| Boston | +1 215 837 0825 | USA |  
| NYC | +1 212 555 3000 | USA |  
| Paris | +33 14 723 4404 | France |  
| Beijing | +86 33 224 5000 | China |  
| Sydney | +61 2 9264 2451 | Australia |  
| London | +44 20 7877 2041 | UK |  
+-----+-----+-----+  
7 rows in set (0.00 sec)
```

在本教程中，我们演示了如何使用 MySQL 命令行将 MySQL 示例数据库导入到 MySQL 数据库服务器中。

在本教程中，您将学习

1. 如何使用 MySQL `SELECT` 语句从表或视图查询数据。
2. 如何使用 MySQL `DISTINCT` 子句与 `SELECT` 语句一起组合来消除结果集中的重复行。

MySQL SELECT 语句简介

使用 `SELECT` 语句从表或视图获取数据。表由行和列组成，如电子表格。通常，我们只希望看到子集行，列的子集或两者的组合。`SELECT` 语句的结果称为结果集，它是行列表，每行由相同数量的列组成。

请参阅示例数据库(`yiibaidb`)中的以下 `employees` 表的结构。它有 8 列：员工人数，姓氏，名字，分机，电子邮件，办公室代码，报告，职位等。

EmployeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1056	Patterson	Mary	x4611	mpatterso@yiibai.com	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@yiibai.com	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson@yiibai.com	6		1056 Sales Manager (APAC)
1102	Bondur	Gerard	x5408	gbondur@gmail.com	4		1056 Sale Manager (EMEA)
1143	Bow	Anthony	x5428	abow@gmail.com	1		1056 Sales Manager (NA)
1165	Jennings	Leslie	x3291	ljennings@yiibai.com	1		1143 Sales Rep
1166	Thompson	Leslie	x4065	lthompson@yiibai.com	1		1143 Sales Rep
1188	Firrelli	Julie	x2173	jfirrelli@yiibai.com	2		1143 Sales Rep
1216	Patterson	Steve	x4334	spatterson@yiibai.com	2		1143 Sales Rep
1286	Tseng	Foon Yue	x2248	ftseng@yiibai.com	3		1143 Sales Rep
1323	Vanauf	George	x4102	gvanauf@yiibai.com	3		1143 Sales Rep
1337	Bondur	Loui	x6493	lbondur@yiibai.com	4		1102 Sales Rep
1370	Hernandez	Gerard	x2028	ghernande@gmail.com	4		1102 Sales Rep
1401	Castillo	Pamela	x2759	pcastillo@gmail.com	4		1102 Sales Rep

`SELECT` 语句控制要查看哪些列和行。例如，如果只对所有员工的名字，姓氏和职位感兴趣，或者您只想查看其职位是销售代表的每位员工的信息，则 `SELECT` 语句可帮助您执行这些操作。

我们来看一下 `SELECT` 语句的语法：

```

SELECT
    column_1, column_2, ...
FROM
    table_1
[INNER | LEFT | RIGHT] JOIN table_2 ON conditions
WHERE
    conditions
GROUP BY column_1
HAVING group_conditions
ORDER BY column_1
LIMIT offset, length;

```

`SELECT` 语句由以下列表中所述的几个子句组成：

- `SELECT` 之后是逗号分隔列或星号(`*`)的列表，表示要返回所有列。
- `FROM` 指定要查询数据的表或视图。
- `JOIN` 根据某些连接条件从其他表中获取数据。
- `WHERE` 过滤结果集中的行。
- `GROUP BY` 将一组行组合成小分组，并对每个小分组应用聚合函数。
- `HAVING` 过滤器基于 `GROUP BY` 子句定义的小分组。
- `ORDER BY` 指定用于排序的列的列表。
- `LIMIT` 限制返回行的数量。

语句中的 `SELECT` 和 `FROM` 语句是必须的，其他部分是可选的。

在随后的教程中将更详细地了解每个子句。在本教程中，我们将重点介绍 `SELECT` 语句的简单形式用法。

MySQL SELECT 语句示例

`SELECT` 语句允许通过在 `SELECT` 子句中指定逗号分隔列的列表来查询表的部分数据。例如，如果要仅查看员工的名字，姓氏和职位，请使用以下查询：

```
SELECT
    lastname, firstname, jobtitle
FROM
    employees;
```

即使员工表中有很多列，`SELECT` 语句只返回表中所有行的三列数据，如下图所示：

```
mysql> SELECT lastname, firstname, jobtitle FROM employees;
+-----+-----+-----+
| lastname | firstname | jobtitle |
+-----+-----+-----+
| Murphy   | Diane    | President |
| Patterson | Mary     | VP Sales  |
| Firrelli  | Jeff     | VP Marketing |
| Patterson | William  | Sales Manager (APAC) |
| Bondur   | Gerard   | Sale Manager (EMEA) |
| Bow       | Anthony  | Sales Manager (NA) |
| Jennings  | Leslie   | Sales Rep   |
| Thompson  | Leslie   | Sales Rep   |
| Firrelli  | Julie    | Sales Rep   |
| Patterson | Steve    | Sales Rep   |
| Tseng     | Foon Yue | Sales Rep   |
| Vanauf    | George   | Sales Rep   |
| Bondur   | Loui     | Sales Rep   |
| Hernandez | Gerard   | Sales Rep   |
| Castillo | Pamela   | Sales Rep   |
| Bott      | Larry    | Sales Rep   |
| Jones     | Barry    | Sales Rep   |
| Fixter   | Andy     | Sales Rep   |
| Marsh     | Peter    | Sales Rep   |
| King      | Tom      | Sales Rep   |
| Nishi     | Mami     | Sales Rep   |
| Kato      | Yoshimi  | Sales Rep   |
| Gerard    | Martin   | Sales Rep   |
+-----+-----+-----+
23 rows in set
```

注意比较以下两个语句返回列有什么区别 -

语句-1

```
SELECT lastname, firstname, jobtitle FROM employees;
```

语句-2

```
SELECT * FROM employees;
```

如果要获取 `employees` 表中所有列的数据，可以列出 `SELECT` 子句中的所有列名，或者只需使用星号(`*`)表示您想要从表的所有列获取数据，如下查询：

```
mysql> SELECT * FROM employees;
+-----+-----+-----+-----+-----+
| employeeNumber | lastName | firstName | extension | email
| officeCode    | reportsTo | jobTitle |
+-----+-----+-----+-----+-----+
|      1002     | Murphy   | Diane    | x5800     | dmurphy@y
|iibai.com      | 1        | NULL     | President |
|      1056     | Patterson | Mary     | x4611     | mpatterso
|@yiibai.com    | 1        |          | 1002      | VP Sales
|      1076     | Firrelli  | Jeff     | x9273     | jfirrelli
|@yiibai.com    | 1        |          | 1002      | VP Marketing
|      1088     | Patterson | William  | x4871     | wpatterson
|@yiibai.com    | 6        |          | 1056      | Sales Manager (APAC)
|      1102     | Bondur   | Gerard   | x5408     | gbondur@g
|mail.com       | 4        |          | 1056      | Sale Manager (EMEA)
|      1143     | Bow       | Anthony  | x5428     | abow@gmai
|l.com          | 1        |          | 1056      | Sales Manager (NA)
|      1165     | Jennings  | Leslie   | x3291     | ljenning
|@yiibai.com    | 1        |          | 1143      | Sales Rep
|      1166     | Thompson  | Leslie   | x4065     | lthompson
|@yiibai.com    | 1        |          | 1143      | Sales Rep
|      1188     | Firrelli  | Julie    | x2173     | jfirrelli
|@yiibai.com    | 2        |          | 1143      | Sales Rep
|      1216     | Patterson | Steve    | x4334     | spatterson
|@yiibai.com    | 2        |          | 1143      | Sales Rep
|      1286     | Tseng     | Foon Yue | x2248     | ftseng@yi
|ibai.com       | 3        |          | 1143      | Sales Rep
|      1323     | Vanauf   | George   | x4102     | gvanauf@y
|iibai.com      | 3        |          | 1143      | Sales Rep
|      1337     | Bondur   | Loui     | x6493     | lbondur@y
|iibai.com      | 4        |          | 1102      | Sales Rep
|      1370     | Hernandez | Gerard   | x2028     | ghernande
|@gmail.com     | 4        |          | 1102      | Sales Rep
|      1401     | Castillo | Pamela   | x2759     | pcastillo
|@gmail.com     | 4        |          | 1102      | Sales Rep
|      1501     | Bott      | Larry    | x2311     | lbott@yi
```

2.1 SELECT 语句

Employee ID	Last Name	First Name	Office Code	Title	Email
1504	Jones	Barry	x102	Sales Rep	bjones@gm
1611	Fixter	Andy	x101	Sales Rep	afixter@y
1612	Marsh	Peter	x102	Sales Rep	pmarsh@yi
1619	King	Tom	x103	Sales Rep	tking@gma
1621	Nishi	Mami	x101	Sales Rep	mnishi@gm
1625	Kato	Yoshimi	x102	Sales Rep	ykato@gma
1702	Gerard	Martin	x2312	Sales Rep	mgerard@g
1102					
23 rows in set					

它返回 `employees` 表中的所有列和行。应该使用星号(`*`)进行测试。建议显式获取数据的列，原因如下：

- 使用星号(`*`)可能会返回不使用的列的数据。它在MySQL数据库服务器和应用程序之间产生不必要的I/O磁盘和网络流量。
- 如果明确指定列，则结果集更可预测并且更易于管理。想象一下，当您使用星号(`*`)并且有人通过添加更多列来更改表格数据时，将会得到一个与预期不同的结果集。
- 使用星号(`*`)可能会将敏感信息暴露给未经授权的用户。

在本教程中，您已经了解并熟悉了 MySQL `SELECT` 语句的用法，并通过 `SELECT` 语句从 MySQL 表中查询数据。

MySQL DISTINCT 子句简介

从表中查询数据时，可能会收到重复的行记录。为了删除这些重复行，可以在 `SELECT` 语句中使用 `DISTINCT` 子句。

`DISTINCT` 子句的语法如下：

```
SELECT DISTINCT  
    columns  
FROM  
    table_name  
WHERE  
    where_conditions;
```

MySQL DISTINCT 示例

下面来看看一个使用 `DISTINCT` 子句从 `employees` 表中选择员工的唯一姓氏 (`lastName`) 的简单示例。

首先，使用 `SELECT` 语句从 `employees` 表中查询员工的姓氏 (`lastName`)，如下所示：

```
SELECT  
    lastname  
FROM  
    employees  
ORDER BY lastname;
```

执行上面查询语句，得到以下结果

```
mysql> SELECT lastname FROM employees ORDER BY lastname;
+-----+
| lastname |
+-----+
| Bondur |
| Bondur |
| Bott    |
| Bow     |
| Castillo |
| Firrelli |
| Firrelli |
| Fixter  |
| Gerard  |
| Hernandez |
| Jennings |
| Jones   |
| Kato    |
| King    |
| Marsh   |
| Murphy  |
| Nishi   |
| Patterson |
| Patterson |
| Patterson |
| Thompson |
| Tseng   |
| Vanauf  |
+-----+
23 rows in set
```

可看到上面结果中，有好些结果是重复的，比如：Bondur，Firrelli等，那如何做到相同的结果只显示一个呢？要删除重复的姓氏，请将 DISTINCT 子句添加到 SELECT 语句中，如下所示：

```
SELECT DISTINCT
    lastname
FROM
    employees
ORDER BY lastname;
```

执行上面查询，得到以下输出结果

```
mysql> SELECT DISTINCT lastname FROM employees ORDER BY lastname;

+-----+
| lastname |
+-----+
| Bondur |
| Bott    |
| Bow     |
| Castillo |
| Firrelli |
| Fixter  |
| Gerard  |
| Hernandez |
| Jennings |
| Jones   |
| Kato    |
| King    |
| Marsh   |
| Murphy  |
| Nishi   |
| Patterson |
| Thompson |
| Tseng   |
| Vanauf  |
+-----+
19 rows in set
```

当使用 `DISTINCT` 子句时，重复的姓氏(`lastname`)在结果集中被消除。

MySQL DISTINCT和NULL值

如果列具有 `NULL` 值，并且对该列使用 `DISTINCT` 子句，MySQL将保留一个 `NULL` 值，并删除其它的 `NULL` 值，因为 `DISTINCT` 子句将所有 `NULL` 值视为相同的值。

例如，在 `customers` 表中，有很多行的州(`state`)列是 `NULL` 值。当使用 `DISTINCT` 子句来查询客户所在的州时，我们将看到唯一的州和 `NULL` 值，如下查询所示：

```
SELECT DISTINCT
    state
FROM
    customers;
```

执行上面查询语句后，输出结果如下 -

```
mysql> SELECT DISTINCT state FROM customers;
+-----+
| state |
+-----+
| NULL  |
| NV    |
| Victoria |
| CA    |
| NY    |
| PA    |
| CT    |
| MA    |
| Osaka |
| BC    |
| Qubec |
| Isle of Wight |
| NSW   |
| NJ    |
| Queensland |
| Co. Cork |
| Pretoria |
| NH    |
| Tokyo |
+-----+
19 rows in set
```

MySQL DISTINCT 在多列上的使用

可以使用具有多个列的 `DISTINCT` 子句。在这种情况下，MySQL 使用所有列的组合来确定结果集中行的唯一性。

例如，要从 `customers` 表中获取城市(`city`)和州(`state`)的唯一组合，可以使用以下查询：

```
SELECT DISTINCT
    state, city
FROM
    customers
WHERE
    state IS NOT NULL
ORDER BY state , city;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT DISTINCT state, city FROM customers WHERE state IS
NOT NULL ORDER BY state ,city;
+-----+-----+
| state | city |
+-----+-----+
| BC   | Tsawassen |
| BC   | Vancouver |
| CA   | Brisbane |
| CA   | Burbank |
| CA   | Burlingame |
| CA   | Glendale |
| CA   | Los Angeles |
| CA   | Pasadena |
| CA   | San Diego |
| CA   | San Francisco |
| CA   | San Jose |
| CA   | San Rafael |
| Co. Cork | Cork |
| CT   | Bridgewater |
| CT   | Glendale |
| CT   | New Haven |
| Isle of Wight | Cowes |
| MA   | Boston |
| MA   | Brickhaven |
```

2.2 SELECT DISTINCT 语句

MA	Cambridge
MA	New Bedford
NH	Nashua
NJ	Newark
NSW	Chatswood
NSW	North Sydney
NV	Las Vegas
NY	NYC
NY	White Plains
Osaka	Kita ku
PA	Allentown
PA	Philadelphia
Pretoria	Hatfield
Qubec	Montral
Queensland	South Brisbane
Tokyo	Minato ku
Victoria	Glen Waverly
Victoria	Melbourne

37 rows in set

没有 DISTINCT 子句，将查询获得州(state)和城市(city)的重复组合如下：

```
SELECT
    state, city
FROM
    customers
WHERE
    state IS NOT NULL
ORDER BY state, city;
```

执行上面查询，得到以下结果 -

	state	city
	BC	Tsawassen
	BC	Vancouver
	CA	Brisbane
	CA	Burbank
	CA	Burlingame
	CA	Glendale
	CA	Los Angeles
	CA	Pasadena
	CA	San Diego
	CA	San Francisco
	CA	San Francisco
	CA	San Jose
	CA	San Rafael
	Co. Cork	Cork
	CT	Bridgewater
	CT	Glendale

DISTINCT 子句与 GROUP BY 子句比较

如果在 `SELECT` 语句中使用 `GROUP BY` 子句，而不使用聚合函数，则 `GROUP BY` 子句的行为与 `DISTINCT` 子句类似。

以下语句使用 `GROUP BY` 子句来选择 `customers` 表中客户的唯一 `state` 列的值。

```
SELECT
    state
FROM
    customers
GROUP BY state;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT state FROM customers GROUP BY state;
+-----+
| state |
+-----+
| NULL  |
| BC    |
| CA    |
| Co. Cork |
| CT    |
| Isle of Wight |
| MA    |
| NH    |
| NJ    |
| NSW   |
| NV    |
| NY    |
| Osaka |
| PA    |
| Pretoria |
| Qubec |
| Queensland |
| Tokyo |
| Victoria |
+-----+
19 rows in set
```

可以通过使用 `DISTINCT` 子句来实现类似的结果：

```
mysql> SELECT DISTINCT state FROM customers;
+-----+
| state |
+-----+
| NULL  |
| NV    |
| Victoria |
| CA    |
| NY    |
| PA    |
| CT    |
| MA    |
| Osaka |
| BC    |
| Qubec |
| Isle of Wight |
| NSW   |
| NJ    |
| Queensland |
| Co. Cork |
| Pretoria |
| NH    |
| Tokyo  |
+-----+
19 rows in set
```

一般而言，`DISTINCT` 子句是 `GROUP BY` 子句的特殊情况。`DISTINCT` 子句和 `GROUP BY` 子句之间的区别是 `GROUP BY` 子句可对结果集进行排序，而 `DISTINCT` 子句不进行排序。

如果将 `ORDER BY` 子句添加到使用 `DISTINCT` 子句的语句中，则结果集将被排序，并且与使用 `GROUP BY` 子句的语句返回的结果集相同。

```
SELECT DISTINCT
    state
FROM
    customers
ORDER BY state;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT DISTINCT state FROM customers ORDER BY state;
+-----+
| state |
+-----+
| NULL  |
| BC    |
| CA    |
| Co. Cork |
| CT    |
| Isle of Wight |
| MA    |
| NH    |
| NJ    |
| NSW   |
| NV    |
| NY    |
| Osaka |
| PA    |
| Pretoria |
| Qubec |
| Queensland |
| Tokyo |
| Victoria |
+-----+
19 rows in set
```

MySQL DISTINCT 和聚合函数

可以使用具有聚合函数(例如SUM，AVG和COUNT)的 DISTINCT 子句中，在 MySQL将聚合函数应用于结果集之前删除重复的行。

例如，要计算美国客户的唯一 state 列的值，可以使用以下查询：

```

SELECT
    COUNT(DISTINCT state)
FROM
    customers
WHERE
    country = 'USA';

```

执行上面查询，得到以下结果 -

```

mysql> SELECT COUNT(DISTINCT state) FROM customers WHERE country = 'USA';
+-----+
| COUNT(DISTINCT state) |
+-----+
|          8           |
+-----+
1 row in set

```

MySQL DISTINCT与LIMIT子句

如果要将 DISTINCT 子句与 LIMIT 子句一起使用，MySQL 会在查找 LIMIT 子句中指定的唯一行数时立即停止搜索。

以下查询 customers 表中的前 3 个非空(NOT NULL)唯一 state 列的值。

```

mysql> SELECT DISTINCT state FROM customers WHERE state IS NOT NULL LIMIT 3;
+-----+
| state |
+-----+
| NV   |
| Victoria |
| CA   |
+-----+
3 rows in set

```

在本教程中，我们学习了使用 MySQL DISTINCT 子句的各种方法，例如消除重复行和计数非 NULL 值。

第三节 过滤数据

在本教程中，我们将学习如何在 `SELECT` 语句中使用 MySQL `WHERE` 子句来过滤结果集中的行记录。

MySQL WHERE子句简介

如果使用 `SELECT` 语句 但不使用 `WHERE` 子句 在表中查询数据，则会获取表中的所有行记录，这些行记录中大部分是不想要的行记录。例如，在一些表中存放商业交易中的数据。从这些表中获取所有行，尤其是对于诸如员工，销售订单，采购订单，生产订单等的大型表格来说，这是没有意义的，因为我们经常想要的是一些特定的数据，例如本季度的销售额，今年销量比去年同期的销量等等。

`WHERE` 子句允许根据指定的过滤表达式或条件来指定要选择的行。

您还将学习如何使用 `LIMIT` 子句 来限制 `SELECT` 语句返回的行数。

MySQL WHERE子句示例

我们将继续使用 [示例数据库\(yiibaidb\)](#) 中 `employees` 表中的数据，如下图所示。

假设只想从 `employees` 表中获取销售代表员工，可使用以下查询：

```
SELECT
    lastname, firstname, jobtitle
FROM
    employees
WHERE
    jobtitle = 'Sales Rep';
```

执行上面查询，得到以下结果 -

```
mysql> SELECT lastname, firstname, jobtitle FROM employees WHERE
   jobtitle = 'Sales Rep';
+-----+-----+-----+
| lastname | firstname | jobtitle |
+-----+-----+-----+
| Jennings | Leslie | Sales Rep |
| Thompson | Leslie | Sales Rep |
| Firrelli | Julie | Sales Rep |
| Patterson | Steve | Sales Rep |
| Tseng | Foon Yue | Sales Rep |
| Vanauf | George | Sales Rep |
| Bondur | Loui | Sales Rep |
| Hernandez | Gerard | Sales Rep |
| Castillo | Pamela | Sales Rep |
| Bott | Larry | Sales Rep |
| Jones | Barry | Sales Rep |
| Fixter | Andy | Sales Rep |
| Marsh | Peter | Sales Rep |
| King | Tom | Sales Rep |
| Nishi | Mami | Sales Rep |
| Kato | Yoshimi | Sales Rep |
| Gerard | Martin | Sales Rep |
+-----+-----+-----+
17 rows in set
```

即使 WHERE 子句出现在语句的末尾，但MySQL会首先使用 WHERE 子句中的表达式来选择匹配的行。它选择具有职位名称为销售代表的行记录。

```
jobtitle = 'Sales Rep';
```

MySQL从 SELECT 子句中的选择列表中选择列。

可以像上面的查询一样形成一个简单的条件，或者是将多个表达式与逻辑运算符(如 AND, OR等)组合在一起的一个非常复杂的例子。例如，要在办公室代码 (officeCode)等于 1 中查找所有销售代表，请使用以下查询：

```

SELECT
    lastname, firstname, jobtitle
FROM
    employees
WHERE
    jobtitle = 'Sales Rep' AND officeCode = 1;

```

执行上面查询后，得到以下结果 -

```

mysql> SELECT lastname, firstname, jobtitle FROM employees WHERE
jobtitle = 'Sales Rep' AND officeCode = 1;
+-----+-----+-----+
| lastname | firstname | jobtitle |
+-----+-----+-----+
| Jennings | Leslie   | Sales Rep |
| Thompson | Leslie   | Sales Rep |
+-----+-----+-----+
2 rows in set

```

下表列出了可用于在 WHERE 子句中形成过滤表达式的比较运算符。

操作符	描述
=	等于，几乎任何数据类型都可以使用它。
<> 或 !=	不等于
<	小于，通常使用数字和日期/时间数据类型。
>	大于，
<=	小于或等于
>=	大于或等于

以下查询使用不等于(!=)运算符来获取不是销售代表的其它所有员工：

3.1 WHERE 子句

```
SELECT
    lastname, firstname, jobtitle
FROM
    employees
WHERE
    jobtitle <> 'Sales Rep';
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT lastname, firstname, jobtitle FROM employees WHERE
jobtitle <> 'Sales Rep';
+-----+-----+-----+
| lastname | firstname | jobtitle |
+-----+-----+-----+
| Murphy   | Diane     | President
| Patterson | Mary      | VP Sales
| Firrelli  | Jeff      | VP Marketing
| Patterson | William   | Sales Manager (APAC)
| Bondur    | Gerard    | Sale Manager (EMEA)
| Bow       | Anthony   | Sales Manager (NA)
+-----+-----+-----+
6 rows in set
```

以下查询将获得办公室代码大于 5 的每位员工：

```
mysql> SELECT lastname, firstname, officeCode FROM employees WHERE
officecode > 5;
+-----+-----+-----+
| lastname | firstname | officeCode |
+-----+-----+-----+
| Patterson | William   | 6
| Bott      | Larry     | 7
| Jones     | Barry     | 7
| Fixter    | Andy      | 6
| Marsh     | Peter     | 6
| King      | Tom       | 6
+-----+-----+-----+
6 rows in set
```

3.1 WHERE 子句

办公室代码小于或等于 4 (≤ 4) 的员工呢？

```
SELECT
    lastname, firstname, officeCode
FROM
    employees
WHERE officecode <= 4;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT lastname, firstname, officeCode FROM employees WHERE officecode <= 4;
+-----+-----+-----+
| lastname | firstname | officeCode |
+-----+-----+-----+
| Murphy   | Diane     | 1          |
| Patterson | Mary      | 1          |
| Firrelli  | Jeff      | 1          |
| Bondur   | Gerard    | 4          |
| Bow       | Anthony   | 1          |
| Jennings  | Leslie    | 1          |
| Thompson  | Leslie    | 1          |
| Firrelli  | Julie     | 2          |
| Patterson | Steve     | 2          |
| Tseng     | Foon Yue  | 3          |
| Vanauf    | George    | 3          |
| Bondur   | Loui      | 4          |
| Hernandez | Gerard    | 4          |
| Castillo | Pamela    | 4          |
| Gerard    | Martin    | 4          |
+-----+-----+-----+
15 rows in set
```

更多关于**MySQL WHERE子句**...

还有一些有用的运算符可以在 **WHERE 子句** 中使用来形成复杂的条件，例如：

- **BETWEEN** 选择在给定范围值内的值。
- **LIKE** 匹配基于模式匹配的值。
- **IN** 指定值是否匹配列表中的任何值。

3.1 WHERE 子句

- `IS NULL` 检查该值是否为 `NULL`。

在本教程中，我们学习了如何使用MySQL `WHERE` 子句来根据条件过滤行记录。

在本教程中，将学习如何使用 MySQL AND 运算符组合多个布尔表达式以形成多个条件来过滤数据。

MySQL AND 运算符简介

AND 运算符是组合两个或多个布尔表达式的逻辑运算符，只有当两个表达式求值为 true 时才返回 true。如果两个表达式中的一个求值为 false，则 AND 运算符返回 false。

```
WHERE boolean_expression_1 AND boolean_expression_2
```

以下说明 AND 运算符组合 true，false 和 null 时的结果。

-	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

AND 运算符通常用在 SELECT，UPDATE，DELETE 语句的 WHERE 子句中以形成布尔表达式。AND 运算符也用于 INNER JOIN 或 LEFT JOIN 子句的连接条件。

当求值具有 AND 运算符的表达式时，MySQL 会计算表达式的其余部分，直到可以确定结果为止。该功能称为短路求值。请参见以下示例。

```
SELECT 1 = 0 AND 1 / 0 ;
```

执行上面查询时，得到以下结果 -

```
mysql> SELECT 1 = 0 AND 1 / 0 ;
+-----+
| 1 = 0 AND 1 / 0 |
+-----+
|          0      |
+-----+
1 row in set
```

请注意，在 MySQL 中，`0` 被认为是 `false`，非零被视为 `true`。

MySQL 只计算表达式 `1 = 0 AND 1/0` 的第一部分 `1 = 0`，因为表达式 `1 = 0` 返回 `false`，所以 MySQL 得出结论：整个表达式的结果是 `false`。MySQL 不对表达式的剩余部分求值，即不对 `1/0` 进行求值；如果对 `1/0` 进行求值，它将发出一个错误消息，因为除以零错误。

MySQL AND 运算符示例

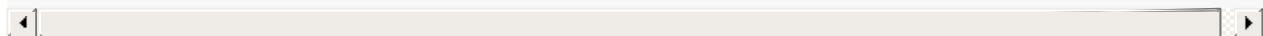
下面使用示例数据库中的 `customers` 表进行演示。`customers` 表的结构如下所示 -

```
mysql> desc customers;
+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default |
| Extra          |            |       |      |          |
+-----+-----+-----+-----+
| customerNumber | int(11)    | NO   | PRI | NULL    |
| customerName   | varchar(50) | NO   |      | NULL    |
| contactLastName | varchar(50) | NO   |      | NULL    |
| contactFirstName | varchar(50) | NO   |      | NULL    |
| phone          | varchar(50) | NO   |      | NULL    |
| addressLine1    | varchar(50) | NO   |      | NULL    |
| addressLine2    | varchar(50) | YES  |      | NULL    |
| city            | varchar(50) | NO   |      | NULL    |
| state           | varchar(50) | YES  |      | NULL    |
| postalCode      | varchar(15) | YES  |      | NULL    |
| country          | varchar(50) | NO   |      | NULL    |
| salesRepEmployeeNumber | int(11) | YES  | MUL | NULL    |
| creditLimit     | decimal(10,2) | YES  |      | NULL    |
+-----+-----+-----+-----+
13 rows in set
```

以下声明选择国家是 USA 和 CA 的客户。我们在 WHERE 子句中使用 AND 运算符。

3.2 AND 运算符

```
SELECT customername, country, state FROM customers WHERE country = 'USA' AND state = 'CA';
```



执行上面查询得到以下结果 -

```
mysql> SELECT customername, country, state FROM customers WHERE country = 'USA' AND state = 'CA';
+-----+-----+-----+
| customername | country | state |
+-----+-----+-----+
| Mini Gifts Distributors Ltd. | USA     | CA    |
| Mini Wheels Co. | USA     | CA    |
| Technics Stores Inc. | USA     | CA    |
| Toys4GrownUps.com | USA     | CA    |
| Boards & Toys Co. | USA     | CA    |
| Collectable Mini Designs Co. | USA     | CA    |
| Corporate Gift Ideas Co. | USA     | CA    |
| Men 'R' US Retailers, Ltd. | USA     | CA    |
| The Sharp Gifts Warehouse | USA     | CA    |
| West Coast Collectables Co. | USA     | CA    |
| Signal Collectibles Ltd. | USA     | CA    |
+-----+-----+-----+
11 rows in set
```

使用 AND 运算符，可以组合两个以上的布尔表达式。例如，以下查询返回位于美国加州的客户，并且信用额度大于 100K。

```
SELECT customername,
       country,
       state,
       creditlimit
  FROM customers
 WHERE country = 'USA'
   AND state = 'CA'
   AND creditlimit > 100000;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT customername, country, state, creditlimit FROM customers
WHERE country = 'USA' AND state = 'CA' AND creditlimit > 100000;
+-----+-----+-----+-----+
| customername | country | state | creditlimit |
+-----+-----+-----+-----+
| Mini Gifts Distributors Ltd. | USA     | CA      | 210500    |
| Collectable Mini Designs Co. | USA     | CA      | 105000    |
| Corporate Gift Ideas Co.   | USA     | CA      | 105000    |
+-----+-----+-----+-----+
3 rows in set
```

在本教程中，我们向您展示了如何使用 MySQL `AND` 运算符组合两个或多个表达式以形成 `WHERE` 子句的复合条件语句。

本教程将学习如何使用MySQL `OR` 运算符组合布尔表达式来过滤数据。

MySQL OR运算符介绍

MySQL `OR` 运算符组合了两个或两个以上布尔表达式。当任一条件为真时，返回 `true`。

下面说明了 `OR` 运算符的语法。

```
boolean_expression_1 OR boolean_expression_2
```

`boolean_expression_1` 和 `boolean_expression_2` 是布尔表达式，它可能返回的结果是：`true`，`false` 或 `NULL`。

下表显示了 `OR` 运算符的结果。

—	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

MySQL OR短路求值

MySQL使用 `OR` 运算符进行短路评估(求值计算)。换句话说，当MySQL可以确定结果时，MySQL会停止评估(求值计算)语句的其余部分。

请参见以下示例。

```
SELECT 1 = 1 OR 1 / 0;
```

执行上面代码，得到以下结果 -

```
mysql> SELECT 1 = 1 OR 1 / 0;
+-----+
| 1 = 1 OR 1 / 0 |
+-----+
|          1      |
+-----+
1 row in set
```

因为表达式 `1 = 1` 总是返回 `true`，MySQL 不会对 `1/0` 进行求值。如果是这样，它会发出一个除以零错误的错误消息。

运算符优先级

当您在语句中使用多个逻辑运算符时，MySQL会在 `AND` 运算符之后再对 `OR` 运算符进行求值。这称为运算符优先级。

运算符优先级决定运算符的求值顺序。MySQL首先对优先级较高的运算符进行求值。

请参见以下示例。

```
SELECT true OR false AND false;
```

执行上面查询，得到如下结果 -

```
mysql> SELECT true OR false AND false;
+-----+
| true OR false AND false |
+-----+
|          1      |
+-----+
1 row in set
```

上面得出的结果，运算的过程是怎么样呢？

- 首先，MySQL 对 `AND` 运算符求值，因此，`false AND false` 返回 `false`。

- 其次，MySQL对 OR 运算符求值，根据返回的 false 值再执行 AND 运算，因此 true OR false 返回 true 。

要更改评估/求值的顺序，请使用括号，例如：

```
SELECT (true OR false) AND false;
```

执行上面查询，得到如下结果 -

```
mysql> SELECT (true OR false) AND false;
+-----+
| (true OR false) AND false |
+-----+
|          0          |
+-----+
1 row in set
```

上面得出的结果，运算的过程是怎么样呢？

- 首先，MySQL计算小括号中的表达式(true OR false)返回 true
- 第二，MySQL评估求值语句的剩余部分，将上面第一步中计算出的表达式结果- true 和剩余部分求值，即： true AND false 返回 false 。

MySQL OR 运算符示例

下面，我们将使用[示例数据库\(yiibaidb\)](#)中的 customers 表进行演示。 customers 表的结果如下所示 -

3.3 OR 运算符

```
mysql> desc customers;
+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default |
| Extra          |            |       |      |          |
+-----+-----+-----+-----+
| customerNumber | int(11)    | NO   | PRI | NULL    |
| customerName   | varchar(50) | NO   |      | NULL    |
| contactLastName | varchar(50) | NO   |      | NULL    |
| contactFirstName | varchar(50) | NO   |      | NULL    |
| phone          | varchar(50) | NO   |      | NULL    |
| addressLine1   | varchar(50) | NO   |      | NULL    |
| addressLine2   | varchar(50) | YES  |      | NULL    |
| city           | varchar(50) | NO   |      | NULL    |
| state          | varchar(50) | YES  |      | NULL    |
| postalCode     | varchar(15) | YES  |      | NULL    |
| country         | varchar(50) | NO   |      | NULL    |
| salesRepEmployeeNumber | int(11)    | YES  | MUL | NULL    |
| creditLimit    | decimal(10,2) | YES  |      | NULL    |
+-----+-----+-----+-----+
13 rows in set
```

例如，要获得美国(USA)或者法国(France)的客户，请在WHERE子句中使用OR运算符，如下所示：

3.3 OR 运算符

```
SELECT
    customername, country
FROM
    customers
WHERE
    country = 'USA' OR country = 'France';
```

执行上面代码，得到如下结果 -

```
mysql> SELECT customername, country FROM customers WHERE country = 'USA' OR country = 'France';
+-----+-----+
| customername | country |
+-----+-----+
| Atelier graphique | France |
| Signal Gift Stores | USA |
| La Rochelle Gifts | France |
| Mini Gifts Distributors Ltd. | USA |
| Mini Wheels Co. | USA |
| Land of Toys Inc. | USA |
| Saveley & Henriot, Co. | France |
| Muscle Machine Inc | USA |
| Diecast Classics Inc. | USA |
| Technics Stores Inc. | USA |
|
+-----+-----+
48 rows in set
```

以下声明返回位于美国(USA)或者法国(France)，并且信用额度大于 10000 的客户。

```
SELECT
    customername, country, creditLimit
FROM
    customers
WHERE
    (country = 'USA' OR country = 'France')
    AND creditlimit > 100000;
```

3.3 OR 运算符

执行上面代码，得到如下结果 -

```
mysql> SELECT customername, country, creditLimit FROM customers
WHERE (country = 'USA' OR country = 'France') AND creditlimit >
100000;
+-----+-----+-----+
| customername | country | creditLimit |
+-----+-----+-----+
| La Rochelle Gifts | France | 118200
| Mini Gifts Distributors Ltd. | USA | 210500
| Land of Toys Inc. | USA | 114900
| Saveley & Henriot, Co. | France | 123900
| Muscle Machine Inc | USA | 138500
| Diecast Classics Inc. | USA | 100600
| Collectable Mini Designs Co. | USA | 105000
| Marta's Replicas Co. | USA | 123700
| Mini Classics | USA | 102700
| Corporate Gift Ideas Co. | USA | 105000
| Online Diecast Creations Co. | USA | 114200
+-----+-----+-----+
11 rows in set
```

请注意，如果不使用括号，查询将返回位于美国的客户或者位于法国并且信用额度大于 10000 的客户。

```
SELECT
    customername, country, creditLimit
FROM
    customers
WHERE
    country = 'USA' OR country = 'France' AND creditlimit > 1000
00;
```

执行上面代码，得到如下结果(共 38 行) -

```
mysql> SELECT customername, country, creditLimit FROM customers
WHERE country = 'USA' OR country = 'France' AND creditlimit > 10
0000;
+-----+-----+-----+
```

3.3 OR 运算符

customername	country	creditLimit
Signal Gift Stores	USA	71800
La Rochelle Gifts	France	118200
Mini Gifts Distributors Ltd.	USA	210500
Mini Wheels Co.	USA	64600
Land of Toys Inc.	USA	114900
Saveley & Henriot, Co.	France	123900
Muscle Machine Inc	USA	138500
Diecast Classics Inc.	USA	100600
Technics Stores Inc.	USA	84600
American Souvenirs Inc	USA	0
Cambridge Collectables Co.	USA	43400
Gift Depot Inc.	USA	84300
Vitachrome Inc.	USA	76400
Auto Moto Classics Inc.	USA	23000
Online Mini Collectables	USA	68700
Toys4GrownUps.com	USA	90700
Boards & Toys Co.	USA	11000
Collectable Mini Designs Co.	USA	105000
Marta's Replicas Co.	USA	123700
Mini Classics	USA	102700
Mini Creations Ltd.	USA	94500
Corporate Gift Ideas Co.	USA	105000
Tekni Collectables Inc.	USA	43000
Classic Gift Ideas, Inc	USA	81100
Men 'R' US Retailers, Ltd.	USA	57700
Gifts4AllAges.com	USA	41900
Online Diecast Creations Co.	USA	114200
Collectables For Less Inc.	USA	70700
Classic Legends Inc.	USA	67500
Gift Ideas Corp.	USA	49700
The Sharp Gifts Warehouse	USA	77600
Super Scale Inc.	USA	95400
Microscale Inc.	USA	39800
FunGiftIdeas.com	USA	85800
West Coast Collectables Co.	USA	55400
Motor Mint Distributors Inc.	USA	72600
Signal Collectibles Ltd.	USA	60300
Diecast Collectables	USA	85100

38 rows in set

在本教程中，您已经学习了如何使用MySQL `OR` 运算符来组合布尔表达式来过滤数据。需要注意的是：使用组合运算符时，`OR` 运算符和 `AND` 运算符的求值顺序。

在本教程中，您将学习如何使用 MySQL `IN` 运算符来确定指定列的值是否匹配列表中的值或子查询中的任何值。

MySQL IN 操作符介绍

`IN` 运算符允许您确定指定的值是否与列表中的值或子查询中的任何值匹配。下面说明了 `IN` 操作符的语法。

```
SELECT
    column1, column2, ...
FROM
    table_name
WHERE
    (expr | column_1) IN ('value1', 'value2', ...);
```

下面我们更详细的来看看上面的查询：

- 可以在 `WHERE` 子句中与 `IN` 运算符一起使用，可使用列或表达式(`expr`)。
- 列表中的值必须用逗号(，)分隔。
- `IN` 操作符也可以用在其他语句(如`INSERT`，`UPDATE`，`DELETE`等)的 `WHERE` 子句中。

如果 `column_1` 的值或 `expr` 表达式的结果等于列表中的任何值，则 `IN` 运算符返回 `1`，否则返回 `0`。

当列表中的值都是常量时：

- 首先，MySQL 根据 `column_1` 的类型或 `expr` 表达式的结果来计算值。
- 第二步，MySQL 排序值。
- 第三步，MySQL 使用二进制搜索算法搜索值。因此，使用具有常量列表的 `IN` 运算符的查询将执行得非常快。

请注意，如果列表中的 `expr` 或任何值为 `NULL`，则 `IN` 运算符计算结果返回 `NULL`。

可以将 `IN` 运算符与 `NOT` 运算符组合，以确定值是否与列表或子查询中的任何值不匹配。

MySQL IN 示例

下面练习一些使用 IN 操作符的例子。首先来看看办事处表：offices 的结构

```
mysql> desc offices;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| officeCode | varchar(10) | NO | PRI | NULL |
| city | varchar(50) | NO | | NULL |
| phone | varchar(50) | NO | | NULL |
| addressLine1 | varchar(50) | NO | | NULL |
| addressLine2 | varchar(50) | YES | | NULL |
| state | varchar(50) | YES | | NULL |
| country | varchar(50) | NO | | NULL |
| postalCode | varchar(15) | NO | | NULL |
| territory | varchar(10) | NO | | NULL |
+-----+-----+-----+-----+-----+
9 rows in set
```

如果您想查找位于美国和法国的办事处，可以使用 IN 运算符作为以下查询：

```
SELECT
    officeCode, city, phone, country
FROM
    offices
WHERE
    country IN ('USA', 'France');
```

执行上面查询语句，得到以下结果

```
mysql> SELECT officeCode, city, phone, country FROM offices WHERE
    country IN ('USA' , 'France');
+-----+-----+-----+-----+
| officeCode | city      | phone    | country |
+-----+-----+-----+-----+
| 1          | San Francisco | 1 650 219 4782 | USA
| 2          | Boston       | 1 215 837 0825 | USA
| 3          | NYC         | 1 212 555 3000 | USA
| 4          | Paris        | 33 14 723 4404 | France
+-----+-----+-----+-----+
4 rows in set
```

也可以使用 `OR` 运算符执行得到与上面查询相同的结果，如下所示：

```
SELECT
    officeCode, city, phone
FROM
    offices
WHERE
    country = 'USA' OR country = 'France';
```

执行上面查询语句，得到以下结果

```
mysql> SELECT officeCode, city, phone FROM offices WHERE country
= 'USA' OR country = 'France';
+-----+-----+-----+
| officeCode | city      | phone    |
+-----+-----+-----+
| 1          | San Francisco | 1 650 219 4782 |
| 2          | Boston       | 1 215 837 0825 |
| 3          | NYC         | 1 212 555 3000 |
| 4          | Paris        | 33 14 723 4404 |
+-----+-----+-----+
4 rows in set
```

如果列表中有很多值，使用多个 `OR` 运算符则会构造一个非常长的语句。因此，使用 `IN` 运算符则会缩短查询并使查询更易读。

要获得不在美国和法国的办事处，请在 WHERE 子句中使用 NOT IN 如下：

```
SELECT
    officeCode, city, phone
FROM
    offices
WHERE
    country NOT IN ('USA', 'France');
```

执行上面查询语句，得到以下结果

```
mysql> SELECT officeCode, city, phone FROM offices WHERE country
NOT IN( 'USA', 'France');
+-----+-----+-----+
| officeCode | city      | phone          |
+-----+-----+-----+
| 5          | Beijing   | +86 33 224 5000
| 6          | Sydney    | +61 2 9264 2451
| 7          | London    | +44 20 7877 2041
+-----+-----+-----+
3 rows in set
```

MySQL IN 与子查询

IN 运算符通常用于子查询。子查询不提供常量值列表，而是提供值列表。

我们来看看两张表：orders 和 orderDetails 表的结构以及它们之间的关系：



例如，如果要查找总金额大于 60000 的订单，则使用 IN 运算符查询如下所示：

```

SELECT
    orderNumber, customerNumber, status, shippedDate
FROM
    orders
WHERE
    orderNumber IN (SELECT
                      orderNumber
                  FROM
                      orderDetails
                  GROUP BY orderNumber
                  HAVING SUM(quantityOrdered * priceEach) > 60000);

```

执行上面语句，得到以下结果

```

mysql> SELECT
        orderNumber, customerNumber, status, shippedDate
    FROM
        orders
    WHERE
        orderNumber IN (SELECT
                          orderNumber
                      FROM
                          orderDetails
                      GROUP BY orderNumber
                      HAVING SUM(quantityOrdered * priceEach) > 60000);
+-----+-----+-----+-----+
| orderNumber | customerNumber | status | shippedDate |
+-----+-----+-----+-----+
| 10165 | 148 | Shipped | 2013-12-26 |
| 10287 | 298 | Shipped | 2014-09-01 |
| 10310 | 259 | Shipped | 2014-10-18 |
+-----+-----+-----+-----+
3 rows in set

```

上面的整个查询可以分为 2 个查询。

首先，子查询使用 `orderDetails` 表中的 `GROUP BY` 和 `HAVING` 子句 返回总额大于 `60000` 的订单号列表。

```

SELECT
    orderNumber
FROM
    orderDetails
GROUP BY orderNumber
HAVING SUM(quantityOrdered * priceEach) > 60000;

```

执行上面语句，得到以下结果

```

mysql> SELECT
    orderNumber
FROM
    orderDetails
GROUP BY orderNumber
HAVING SUM(quantityOrdered * priceEach) > 60000;
+-----+
| orderNumber |
+-----+
| 10165      |
| 10287      |
| 10310      |
+-----+
3 rows in set

```

其次，主查询从 `orders` 表中获取数据，并在 `WHERE` 子句中应用 `IN` 运算符。

```

SELECT
    orderNumber, customerNumber, status, shippedDate
FROM
    orders
WHERE
    orderNumber IN (10165, 10287, 10310);

```

执行上面语句，得到以下结果

```
mysql> SELECT
    orderNumber, customerNumber, status, shippedDate
  FROM
    orders
 WHERE
    orderNumber IN (10165, 10287, 10310);
+-----+-----+-----+-----+
| orderNumber | customerNumber | status | shippedDate |
+-----+-----+-----+-----+
| 10165      |        148 | Shipped | 2013-12-26 |
| 10287      |        298 | Shipped | 2014-09-01 |
| 10310      |        259 | Shipped | 2014-10-18 |
+-----+-----+-----+-----+
3 rows in set
```

在本教程中，我们向您展示了如何使用 MySQL `IN` 运算符来确定值是否匹配列表或子查询中的任何值。

在本教程中，您将学习如何使用 MySQL BETWEEN 运算符，使用它来确定值是否在一个值范围内。

MySQL BETWEEN 运算符介绍

BETWEEN 运算符允许指定要测试的值范围。我们经常在 **SELECT**，**INSERT**，**UPDATE** 和 **DELETE** 语句的 **WHERE 子句** 中使用 BETWEEN 运算符。

下面说明了 BETWEEN 运算符的语法：

```
expr [NOT] BETWEEN begin_expr AND end_expr;
```

expr 是在由 begin_expr 和 end_expr 定义的范围内测试的表达式。

所有三个表达式：expr，begin_expr 和 end_expr 必须具有相同的 **数据类型**。

如果 expr 的值大于或等于(\geq) begin_expr 的值且小于等于(\leq) end_expr 的值，则 BETWEEN 运算符返回 true，否则返回 0。

如果 expr 的值小于($<$) begin_expr 的值或大于 end_expr 的值，则 NOT BETWEEN 将返回 true，否则返回 0。

如果任何表达式为 NULL，则 BETWEEN 运算符返回 NULL 值。如果想指定一个不含边界值的范围，则使用大于($>$)和小于($<$)运算符。

MySQL BETWEEN示例

下面我们来练习一些使用 BETWEEN 运算符的例子。

MySQL BETWEEN与数字示例

请参见[示例数据库\(yiibaidb\)](#)中的以下产品(products)表，表的结构如下所示：

```
mysql> desc products;
+-----+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default | Ex |
+-----+-----+-----+-----+-----+
| productCode    | varchar(15) | NO   | PRI | NULL    |    |
| productName     | varchar(70)  | NO   |      | NULL    |    |
| productLine     | varchar(50)  | NO   | MUL | NULL    |    |
| productScale    | varchar(10)  | NO   |      | NULL    |    |
| productVendor   | varchar(50)  | NO   |      | NULL    |    |
| productDescription | text        | NO   |      | NULL    |    |
| quantityInStock | smallint(6) | NO   |      | NULL    |    |
| buyPrice        | decimal(10,2) | NO   |      | NULL    |    |
| MSRP            | decimal(10,2) | NO   |      | NULL    |    |
+-----+-----+-----+-----+-----+
9 rows in set
```

假设您想要查找价格在 90 和 100 (含 90 和 100)元范围内的商品，可以使用 **BETWEEN** 运算符作为以下查询：

```
mysql> SELECT
    productCode, productName, buyPrice
  FROM
    products
 WHERE
    buyPrice BETWEEN 90 AND 100;
+-----+-----+-----+
| productCode | productName | buyPrice |
+-----+-----+-----+
| S10_1949   | 1952 Alpine Renault 1300 | 98.58   |
| S10_4698   | 2003 Harley-Davidson Eagle Drag Bike | 91.02   |
| S12_1099   | 1968 Ford Mustang           | 95.34   |
| S12_1108   | 2001 Ferrari Enzo           | 95.59   |
| S18_1984   | 1995 Honda Civic            | 93.89   |
| S18_4027   | 1970 Triumph Spitfire        | 91.92   |
| S24_3856   | 1956 Porsche 356A Coupe      | 98.3    |
+-----+-----+-----+
7 rows in set
```

也可以通过使用大于或等于(`>=`)和小于或等于(`<=`)运算符来实现相同的结果，如以下查询：

```
mysql> SELECT
    productCode, productName, buyPrice
  FROM
    products
 WHERE
    buyPrice >= 90 AND buyPrice <= 100;
+-----+-----+-----+
| productCode | productName | buyPrice |
+-----+-----+-----+
| S10_1949   | 1952 Alpine Renault 1300 | 98.58   |
| S10_4698   | 2003 Harley-Davidson Eagle Drag Bike | 91.02   |
| S12_1099   | 1968 Ford Mustang           | 95.34   |
| S12_1108   | 2001 Ferrari Enzo            | 95.59   |
| S18_1984   | 1995 Honda Civic             | 93.89   |
| S18_4027   | 1970 Triumph Spitfire         | 91.92   |
| S24_3856   | 1956 Porsche 356A Coupe       | 98.3    |
+-----+-----+-----+
7 rows in set
```

要查找购买价格不在 20 到 100 (含 20 到 100)之间的产品，可将 BETWEEN 运算符与 NOT 运算符组合使用，如下：

```
mysql> SELECT
    productCode, productName, buyPrice
  FROM
    products
 WHERE
    buyPrice NOT BETWEEN 20 AND 100;
+-----+-----+-----+
| productCode | productName | buyPrice |
+-----+-----+-----+
| S10_4962   | 1962 Lancia Delta 16V | 103.42 |
| S18_2238   | 1998 Chrysler Plymouth Prowler | 101.51 |
| S24_2840   | 1958 Chevy Corvette Limited Edition | 15.91 |
| S24_2972   | 1982 Lamborghini Diablo | 16.24 |
+-----+-----+-----+
4 rows in set
```

您也可以使用少于(`<`)，大于(`>`)和逻辑运算符(`AND`)重写上述查询，如下所示 -

```
SELECT
    productCode, productName, buyPrice
  FROM
    products
 WHERE
    buyPrice < 20 OR buyPrice > 100;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
    productCode, productName, buyPrice
  FROM
    products
 WHERE
    buyPrice < 20 OR buyPrice > 100;
+-----+-----+-----+
| productCode | productName | buyPrice |
+-----+-----+-----+
| S10_4962   | 1962 Lancia Delta 16V | 103.42  |
| S18_2238   | 1998 Chrysler Plymouth Prowler | 101.51  |
| S24_2840   | 1958 Chevy Corvette Limited Edition | 15.91  |
| S24_2972   | 1982 Lamborghini Diablo | 16.24  |
+-----+-----+-----+
4 rows in set
```

MySQL BETWEEN 与日期类型数据示例

当使用 `BETWEEN` 运算符与日期类型值时，要获得最佳结果，应该使用类型转换将列或表达式的类型显式转换为 `DATE` 类型。

例如，要查询获取所需日期(`requiredDate`)从 `2013-01-01` 到 `2013-01-31` 的所有订单，请使用以下查询：

```
SELECT orderNumber,
       requiredDate,
       status
  FROM orders
 WHERE requireddate
       BETWEEN CAST('2013-01-01' AS DATE)
       AND CAST('2013-01-31' AS DATE);
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT orderNumber,
       requiredDate,
       status
  FROM orders
 WHERE requiredDate
   BETWEEN CAST('2013-01-01' AS DATE)
   AND CAST('2013-01-31' AS DATE);
+-----+-----+-----+
| orderNumber | requiredDate | status |
+-----+-----+-----+
| 10100 | 2013-01-13 | Shipped |
| 10101 | 2013-01-18 | Shipped |
| 10102 | 2013-01-18 | Shipped |
+-----+-----+-----+
3 rows in set
```

因为 `requiredDate` 列的数据类型是 `DATE`，所以我们使用转换运算符将文字字符串“`2013-01-01`”和“`2013-12-31`”转换为 `DATE` 数据类型。

在本教程中，您已经学会了如何使用 `BETWEEN` 运算符来测试值是否在值的范围内。

在本教程中，您将了解如何使用 MySQL `LIKE` 运算符根据模式查询选择数据。

`LIKE` 操作符通常用于基于模式查询选择数据。以正确的方式使用 `LIKE` 运算符对于增加/减少查询性能至关重要。

`LIKE` 操作符允许您根据指定的模式从表中查询选择数据。因此，`LIKE` 运算符通常用在 [SELECT语句](#) 的 [WHERE子句](#) 中。

MySQL 提供两个通配符，用于与 `LIKE` 运算符一起使用，它们分别是：百分比符号 - `%` 和下划线 - `_`。

- 百分比(`%`)通配符允许匹配任何字符串的零个或多个字符。
- 下划线(`_`)通配符允许匹配任何单个字符。

MySQL LIKE 运行符示例

下面让我们来学习一些使用 `LIKE` 操作符的例子。请参阅以下 `employees` 表。

```
mysql> desc employees;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| employeeNumber | int(11) | NO | PRI | NULL |
| lastName | varchar(50) | NO | | NULL |
| firstName | varchar(50) | NO | | NULL |
| extension | varchar(10) | NO | | NULL |
| email | varchar(100) | NO | | NULL |
| officeCode | varchar(10) | NO | MUL | NULL |
| reportsTo | int(11) | YES | MUL | NULL |
| jobTitle | varchar(50) | NO | | NULL |
+-----+-----+-----+-----+-----+
8 rows in set
```

MySQL LIKE 使用百分比(%)通配符

假设要搜索名字以字符 `a` 开头的员工信息，可以在模式末尾使用百分比通配符 (`%`)，如下所示：

```

SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    firstName LIKE 'a%';

```

执行上面查询，得到以下结果 -

```

mysql> SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    firstName LIKE 'a%';
+-----+-----+-----+
| employeeNumber | lastName | firstName |
+-----+-----+-----+
|      1143 | Bow      | Anthony   |
|      1611 | Fixter   | Andy      |
+-----+-----+-----+
2 rows in set

```

MySQL将扫描整个 `employees` 表以找到每个其名字以字符 `a` 开头，后跟任意数量的字符的员工信息。

要搜索员工以 `on` 字符结尾的姓氏，例如，`Patterson`，`Thompson`，可以使用模式开头的 `%` 通配符，如下查询：

```

SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    lastName LIKE '%on';

```

执行上面查询，得到以下结果 -

```
mysql> SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    lastName LIKE '%on';
+-----+-----+-----+
| employeeNumber | lastName | firstName |
+-----+-----+-----+
|      1056 | Patterson | Mary
|      1088 | Patterson | William
|     1166 | Thompson | Leslie
|     1216 | Patterson | Steve
+-----+-----+-----+
4 rows in set
```

如果知道要搜索包含指定字符串，则可以在模式的开头和结尾使用百分比(%)通配符。

例如，要查找 lastname 字段值中包含 on 字符串的所有员工，可使用带有 %on% 条件，如下所示

```
SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    lastname LIKE '%on%';
```

执行上面查询，得到以下结果 -

```
mysql> SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    lastname LIKE '%on%';
+-----+-----+-----+
| employeeNumber | lastName | firstName |
+-----+-----+-----+
|      1056 | Patterson | Mary
|      1088 | Patterson | William
|     1102 | Bondur | Gerard
|     1166 | Thompson | Leslie
|     1216 | Patterson | Steve
|     1337 | Bondur | Loui
|     1504 | Jones | Barry
+-----+-----+-----+
7 rows in set
```

MySQL LIKE 带下划线（_）通配符

要查找名字以 T 开头的员工，以 m 结尾，并且包含例如 Tom ， Tim 之间的任何单个字符，可以使用下划线通配符来构建模式，如下所示：

```
SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    firstname LIKE 'T_m';
```

执行上面查询，得到以下结果 -

```
mysql> SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    firstname LIKE 'T_m';
+-----+-----+-----+
| employeeNumber | lastName | firstName |
+-----+-----+-----+
|       1619 | King     | Tom      |
+-----+-----+-----+
1 row in set
```

具有 NOT 运算符的 MySQL LIKE 运算符

MySQL 允许将 NOT 运算符与 LIKE 运算符组合，以找到不匹配特定模式的字符串。

假设要搜索姓氏(lastname)不以字符 B 开头的员工，则可以使用 NOT LIKE 作为以下查询：

```
SELECT
    employeeNumber, lastName, firstName
FROM
    employees
WHERE
    lastName NOT LIKE 'B%';
```

执行上面查询语句，得到以下结果 -

```

mysql> SELECT
    employeeNumber, lastName, firstName
  FROM
    employees
 WHERE
    lastName NOT LIKE 'B%';
+-----+-----+-----+
| employeeNumber | lastName | firstName |
+-----+-----+-----+
|      1002     | Murphy   | Diane    |
|      1056     | Patterson | Mary     |
|      1076     | Firrelli  | Jeff     |
|      1088     | Patterson | William  |
|      1165     | Jennings  | Leslie   |
|      1166     | Thompson  | Leslie   |
|      1188     | Firrelli  | Julie    |
|      1216     | Patterson | Steve    |
|      1286     | Tseng     | Foon Yue |
|      1323     | Vanauf    | George   |
|      1370     | Hernandez | Gerard   |
|      1401     | Castillo  | Pamela   |
|      1504     | Jones     | Barry    |
|      1611     | Fixter    | Andy     |
|      1612     | Marsh     | Peter    |
|      1619     | King      | Tom      |
|      1621     | Nishi     | Mami    |
|      1625     | Kato      | Yoshimi  |
|      1702     | Gerard    | Martin   |
+-----+-----+-----+
19 rows in set

```

请注意，使用 `LIKE` 运算符，该模式不区分大小写，因此，`b%` 和 `B%` 模式产生相同的结果。

MySQL LIKE 与 ESCAPE 子句

有时想要匹配的模式包含通配符，例如 `10%`，`_20` 等这样的字符串时。在这种情况下，您可以使用 `ESCAPE` 子句指定转义字符，以便 MySQL 将通配符解释为文字字符。如果未明确指定转义字符，则反斜杠字符 `\` 是默认转义字符。

3.6 LIKE 运算符

如下语句，将查询 productCode 字段中包含 _20 字符串的值。

```
SELECT
    productCode, productName
FROM
    products
WHERE
    productCode LIKE '%\_20%';
```

执行上面语句，得到以下结果 -

```
mysql> SELECT
    productCode, productName
FROM
    products
WHERE
    productCode LIKE '%\_20%';
+-----+-----+
| productCode | productName
+-----+-----+
| S10_2016   | 1996 Moto Guzzi 1100i
| S24_2000   | 1960 BSA Gold Star DBD34
| S24_2011   | 18th century schooner
| S24_2022   | 1938 Cadillac V 16 Presidential Limousine
| S700_2047  | HMS Bounty
+-----+-----+
5 rows in set
```

或者，也可以使用 ESCAPE 子句指定一个不同的转义字符，例如 \$:

```
SELECT
    productCode, productName
FROM
    products
WHERE
    productCode LIKE '%$_20%' ESCAPE '$';
```

以上语句查询结果与上一个语句得到的结果相同。

模式 `%%_20%` 匹配任何包含 `_20` 字符串的字符串。

`LIKE` 操作符强制 MySQL 扫描整个表以找到匹配的行记录，因此，它不允许数据库引擎使用索引进行快速搜索。因此，当要从具有大量行的表查询数据时，使用 `LIKE` 运算符来查询数据的性能会大幅降低。

在本教程中，您已经学习了如何使用 `LIKE` 运算符根据模式查询数据，这比使用比较运算符更灵活。

在本教程中，您将学习如何使用MySQL `LIMIT` 子句来限制 `SELECT` 语句返回记录的行数。

MySQL LIMIT子句简介

在 `SELECT` 语句中使用 `LIMIT` 子句来约束结果集中的行数。`LIMIT` 子句接受一个或两个参数。两个参数的值必须为零或正[整数](#)。

下面说明了两个参数的 `LIMIT` 子句语法：

```
SELECT  
    column1, column2, ...  
FROM  
    table  
LIMIT offset , count;
```

我们来查看 `LIMIT` 子句参数：

- `offset` 参数指定要返回的第一行的偏移量。第一行的偏移量为 `0`，而不是 `1`。
- `count` 指定要返回的最大行数。



当您使用带有一个参数的 `LIMIT` 子句时，此参数将用于确定从结果集的开头返回的最大行数。

```
SELECT  
    column1, column2, ...  
FROM  
    table  
LIMIT count;
```

上面的查询等同于下面接受两个参数的 `LIMIT` 子句的查询：

```
SELECT  
    column1, column2, ...  
FROM  
    table  
LIMIT 0 , count;
```

使用 MySQL LIMIT 获取前 N 行

可以使用 `LIMIT` 子句来选择表中的前 `N` 行记录，如下所示：

```
SELECT  
    column1, column2, ...  
FROM  
    table  
LIMIT N;
```

例如，要查询 `employees` 表中前 5 个客户，请使用以下查询：

```
SELECT customernumber, customername, creditlimit FROM customers  
LIMIT 5;
```

或者 -

```
SELECT customernumber, customername, creditlimit FROM customers  
LIMIT 0,5;
```

执行上面语句，得到以下结果 -

```
mysql> SELECT customernumber, customername, creditlimit FROM customers LIMIT 5;
+-----+-----+-----+
| customernumber | customername | creditlimit |
+-----+-----+-----+
| 103 | Atelier graphique | 21000 |
| 112 | Signal Gift Stores | 71800 |
| 114 | Australian Collectors, Co. | 117300 |
| 119 | La Rochelle Gifts | 118200 |
| 121 | Baane Mini Imports | 81700 |
+-----+-----+-----+
5 rows in set
```

使用MySQL LIMIT 获得最高和最低的值

LIMIT 子句经常与 ORDER BY 子句一起使用。首先，使用 ORDER BY 子句根据特定条件对结果集进行排序，然后使用 LIMIT 子句来查找最小或最大值。

注意： ORDER BY 子句按指定字段排序的使用。

请参见[示例数据库\(yiibaidb\)](#)中的以下 `customers` 表，其表结构如下所示 -

```
mysql> desc customers;
+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default |
| Extra          |            |       |      |          |
+-----+-----+-----+-----+
| customerNumber | int(11)    | NO   | PRI | NULL    |
| customerName   | varchar(50) | NO   |      | NULL    |
| contactLastName | varchar(50) | NO   |      | NULL    |
| contactFirstName | varchar(50) | NO   |      | NULL    |
| phone          | varchar(50) | NO   |      | NULL    |
| addressLine1   | varchar(50) | NO   |      | NULL    |
| addressLine2   | varchar(50) | YES  |      | NULL    |
| city           | varchar(50) | NO   |      | NULL    |
| state          | varchar(50) | YES  |      | NULL    |
| postalCode     | varchar(15) | YES  |      | NULL    |
| country         | varchar(50) | NO   |      | NULL    |
| salesRepEmployeeNumber | int(11)    | YES  | MUL | NULL    |
| creditLimit    | decimal(10,2) | YES  |      | NULL    |
+-----+-----+-----+-----+
13 rows in set
```

例如，要查询信用额度最高的前五名客户，请使用以下查询：

```
SELECT customernumber, customername, creditlimit
FROM customers
ORDER BY creditlimit DESC
LIMIT 5;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT customernumber, customername, creditlimit
FROM customers
ORDER BY creditlimit DESC
LIMIT 5;
+-----+-----+-----+
| customernumber | customername | creditlimit |
+-----+-----+-----+
| 141 | Euro+ Shopping Channel | 227600
| 124 | Mini Gifts Distributors Ltd. | 210500
| 298 | Vida Sport, Ltd | 141300
| 151 | Muscle Machine Inc | 138500
| 187 | AV Stores, Co. | 136800
+-----+-----+-----+
5 rows in set
```

以下查询将返回信用限额最低的五位客户：

```
SELECT customernumber, customername, creditlimit
FROM customers
ORDER BY
creditlimit ASC
LIMIT 5;
```

使用MySQL LIMIT 获得第n个最高值

MySQL中最棘手的问题之一是：如何获得结果集中的第 n 个最高值，例如查询第二(或第 n)贵的产品是哪个，显然不能使用MAX或MIN这样的函数来查询获得。但是，我们可以使用MySQL LIMIT 来解决这样的问题。

- 首先，按照降序对结果集进行排序。

3.7 LIMIT 子句

- 第二步，使用 `LIMIT` 子句获得第 `n` 贵的产品。

通用查询如下：

```
SELECT
    column1, column2, ...
FROM
    table
ORDER BY column1 DESC
LIMIT nth 1, count;
```

下面我们来看看一个例子，将使用[示例数据库\(yiibaidb\)](#)中的产品(`products`)表来进行演示。`products` 表的结构如下所示 -

```
mysql> desc products;
+-----+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default | Ex |
+-----+-----+-----+-----+-----+
| tra           | varchar(15) | NO   | PRI | NULL    |     |
+-----+-----+-----+-----+-----+
| productCode   | varchar(15) | NO   | PRI | NULL    |     |
| productName   | varchar(70)  | NO   |     | NULL    |     |
| productLine   | varchar(50)  | NO   | MUL | NULL    |     |
| productScale  | varchar(10)  | NO   |     | NULL    |     |
| productVendor | varchar(50)  | NO   |     | NULL    |     |
| productDescription | text      | NO   |     | NULL    |     |
| quantityInStock | smallint(6) | NO   |     | NULL    |     |
| buyPrice      | decimal(10,2) | NO   |     | NULL    |     |
| MSRP          | decimal(10,2) | NO   |     | NULL    |     |
+-----+-----+-----+-----+-----+
9 rows in set
```

查看以下产品表中的行记录：

```
mysql> SELECT productCode, productName, buyprice
   FROM products
  ORDER BY
    buyprice DESC;
+-----+-----+-----+
| productCode | productName | buyprice |
+-----+-----+-----+
| S10_4962   | 1962 Lancia Delta 16V | 103.42  |
| S18_2238   | 1998 Chrysler Plymouth Prowler | 101.51  |
| S10_1949   | 1952 Alpine Renault 1300 | 98.58   |
| S24_3856   | 1956 Porsche 356A Coupe | 98.3    |
| S12_1108   | 2001 Ferrari Enzo | 95.59   |
| S12_1099   | 1968 Ford Mustang | 95.34   |
+-----+-----+-----+
110 rows in set
```

我们的任务找出结果集中价格第二高的产品。可以使用 `LIMIT` 子句来选择第二行，如以下查询(注意：偏移量从 `0` 开始，所以要指定从 `1` 开始，然后取一行记录)：

```
SELECT productCode, productName, buyprice FROM products
ORDER BY buyprice DESC
LIMIT 1, 1;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT productCode, productName, buyprice FROM products  
ORDER BY buyprice DESC  
LIMIT 1, 1;  
+-----+-----+-----+  
| productCode | productName | buyprice |  
+-----+-----+-----+  
| S18_2238 | 1998 Chrysler Plymouth Prowler | 101.51 |  
+-----+-----+-----+  
1 row in set
```

类似的，获取售价第三高、第四高的产品信息为：`LIMIT 2, 1` 和 `LIMIT 3, 1`。

在本教程中，我们向您展示了如何使用MySQL `LIMIT` 子句来限制 `SELECT` 语句返回的行数。通过本教程的学习，相信您应该对MySQL `LIMIT` 子句的使用有一定理解了。

在本教程中，您将学习如何使用 MySQL `IS NULL` 运算符来测试值是否为一个 `NULL` 值。

MySQL IS NULL 操作符简介

要测试值是否为 `NULL` 值，需要使用 `IS NULL` 运算符。以下显示 `IS NULL` 运算符的语法：

```
value IS NULL
```

如果值为 `NULL`，该表达式将返回 `true`。否则返回 `false`。

请注意，MySQL 没有内置的 `BOOLEAN` 类型。它使用 `TINYINT(1)` 来表示 `BOOLEAN` 值，即 `1` 表示 `true`，`0` 表示 `false`。

因为 `IS NULL` 是一个比较运算符，所以您可以在任何使用运算符的地方使用它，例如在 `SELECT` 或 `WHERE` 子句中。如下面的例子：

```
SELECT 1 IS NULL, # -- 0
      0 IS NULL, # -- 0
      NULL IS NULL; # -- 1;
```

要检查值是否不为 `NULL`，请使用 `IS NOT NULL` 运算符，如下所示：

```
value IS NOT NULL
```

如果该值不为 `NULL`，则此表达式返回 `true`（也就是 `1`）。否则返回 `false`（也就是 `0`）。请考虑以下示例：

```
SELECT 1 IS NOT NULL, #-- 1
      0 IS NOT NULL, #-- 1
      NULL IS NOT NULL; #-- 0;
```

上面查询语句，执行后得到以下结果 -

```
mysql> SELECT 1 IS NOT NULL, #-- 1
      0 IS NOT NULL, #-- 1
      NULL IS NOT NULL; #-- 0
+-----+-----+-----+
| 1 IS NOT NULL | 0 IS NOT NULL | NULL IS NOT NULL |
+-----+-----+-----+
|          1 |           1 |          0 |
+-----+-----+-----+
1 row in set
```

MySQL IS NULL 示例

我们将使用示例数据库(yiibaidb)中的 customers 表进行演示，customers 表的结构如下所示 -

```
mysql> desc customers;
+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default |
| Extra          |            |       |      |          |
+-----+-----+-----+-----+
| customerNumber | int(11)    | NO   | PRI | NULL    |
| customerName   | varchar(50) | NO   |      | NULL    |
| contactLastName | varchar(50) | NO   |      | NULL    |
| contactFirstName | varchar(50) | NO   |      | NULL    |
| phone          | varchar(50) | NO   |      | NULL    |
| addressLine1    | varchar(50) | NO   |      | NULL    |
| addressLine2    | varchar(50) | YES  |      | NULL    |
| city            | varchar(50) | NO   |      | NULL    |
| state           | varchar(50) | YES  |      | NULL    |
| postalCode      | varchar(15) | YES  |      | NULL    |
| country          | varchar(50) | NO   |      | NULL    |
| salesRepEmployeeNumber | int(11) | YES  | MUL | NULL    |
| creditLimit     | decimal(10,2) | YES  |      | NULL    |
+-----+-----+-----+-----+
13 rows in set
```

要查询没有销售代表的客户，请使用 `IS NULL` 运算符，如下所示：

```
SELECT
    customerName,
    country,
    salesrepemployeenumber
FROM
    customers
WHERE
    salesrepemployeenumber IS NULL
ORDER BY customerName;
```

执行上面查询，得到以下结果 -

```

mysql> SELECT
    customerName,
    country,
    salesrepemployeenumber
FROM
    customers
WHERE
    salesrepemployeenumber IS NULL
ORDER BY customerName;
+-----+-----+
| customerName | country | salesrepemployeenumber |
+-----+-----+
| ANG Resellers | Spain | NULL
| Anton Designs, Ltd. | Spain | NULL
| Asian Shopping Network, Co | Singapore | NULL
| Asian Treasures, Inc. | Ireland | NULL
| BG&E Collectables | Switzerland | NULL
| Cramer Spezialitten, Ltd | Germany | NULL
| Der Hund Imports | Germany | NULL
| Schuyler Imports | Netherlands | NULL
| Stuttgart Collectable Exchange | Germany | NULL
| Warburg Exchange | Germany | NULL
...
+-----+-----+
22 rows in set

```

3.8 IS NULL 运算符

要查询有销售代表的客户，请使用 `IS NOT NULL` 运算符，如下查询语句 -

```
SELECT
    customerName,
    country,
    salesrepemployeenumber
FROM
    customers
WHERE
    salesrepemployeenumber IS NOT NULL
ORDER BY customerName;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT
    customerName,
    country,
    salesrepemployeenumber
FROM
    customers
WHERE
    salesrepemployeenumber IS NOT NULL
ORDER BY customerName;
+-----+-----+
| customerName | country | salesrepemp
loyeenumber |
+-----+-----+
| Alpha Cognac | France |
| 1370          |         |
| American Souvenirs Inc | USA |
| 1286          |         |
| Amica Models & Co. | Italy |
| 1401          |         |
| Anna's Decorations, Ltd | Australia |
| 1611          |         |
| Atelier graphique | France |
| 1370          |         |
| Australian Collectables, Ltd | Australia |
|
```

1611		
Australian Collectors, Co.	Australia	
1611		
Australian Gift Network, Co	Australia	
1611		
Auto Associs & Cie.	France	
1370		
Auto Canal + Petit	France	
1337		
Auto Moto Classics Inc.	USA	
1216		
AV Stores, Co.	UK	
1501		
Baane Mini Imports	Norway	
1504		
Bavarian Collectables Imports, Co.	Germany	
1504		
.....		
+-----+-----+-----+		
------+-----+		
100 rows in set		

MySQL IS NULL 的专用功能

为了兼容 ODBC 程序，MySQL 支持 IS NULL 运算符的一些专门功能。

(1). 如果具有 NOT NULL 约束的 DATE 或 DATETIME 列包含特殊日期'0000-00-00'，则可以使用 IS NULL 运算符来查找这些行。如下示例 -

```

CREATE TABLE IF NOT EXISTS projects (
    id INT AUTO_INCREMENT,
    title VARCHAR(255),
    begin_date DATE NOT NULL,
    complete_date DATE NOT NULL,
    PRIMARY KEY(id)
);

INSERT INTO projects(title,begin_date, complete_date)
VALUES('New CRM', '2020-01-01', '0000-00-00'),
      ('ERP Future', '2020-01-01', '0000-00-00'),
      ('VR', '2020-01-01', '2030-01-01');

SELECT
    *
FROM
    projects
WHERE
    complete_date IS NULL;

```

在这个例子中，创建了一个 `projects` 新表，并将一些数据插入到表中。最后一个查询使用 `IS NULL` 来获取 `complete_date` 列中的值为“`0000-00-00`”的行。

(2). 如果变量 `@@sql_auto_is_null` 设置为 `1`，则可以使用 `IS NULL` 运算符在执行 `INSERT` 语句后获取生成列的值。请注意，默认情况下，变量 `@@sql_auto_is_null` 为 `0`。请参见以下示例。

首先，将变量 `@@sql_auto_is_null` 设置为 `1`。

```
SET @@sql_auto_is_null = 1;
```

第二步，在 `projects` 表中插入一个新行：

```

INSERT INTO projects(title,begin_date, complete_date)
VALUES('MRP III', '2010-01-01', '2020-12-31');

```

第三步，使用 `IS NULL` 运算符来获取 `id` 列的生成值：

```

SELECT
    id
FROM
    projects
WHERE
    id IS NULL;

```

MySQL IS NULL 优化

MySQL 对于 `IS NULL` 运算符执行相同的优化方式与等于(`=`)运算符相同。

例如，MySQL 在使用 `IS NULL` 运算符搜索 `NULL` 时使用索引，如以下查询所示：

```

SELECT
    customerNumber,
    salesRepEmployeeNumber
FROM
    customers
WHERE
    salesRepEmployeeNumber IS NULL;

```

查看 `EXPLAIN` 查询过程：

```

EXPLAIN SELECT
    customerNumber,
    salesRepEmployeeNumber
FROM
    customers
WHERE
    salesRepEmployeeNumber IS NULL;

```

执行上面查询语句，输出以下结果 -

```
mysql> EXPLAIN SELECT
    customerNumber,
    salesRepEmployeeNumber
  FROM
    customers
 WHERE
    salesRepEmployeeNumber IS NULL;
+-----+-----+-----+-----+
| id  | select_type | table   | partitions | type  | possible_keys | |
|     |             | key      |            |         | ref    | rows       |
|     |             | filtered |             |         | const  |             |
|     |             | Extra    |             |         |          |             |
+-----+-----+-----+-----+
| 1   | SIMPLE      | customers | NULL      | ref   | salesRepEmp | |
|      |             |           | salesRepEmployeeNumber | 5     | const  | 22          |
|      |             |           |             | 100   | Using where; Using index |
+-----+-----+-----+-----+
1 row in set
```

MySQL 也可以优化组合 `col = value OR col IS NULL`。请参阅以下示例：

```
EXPLAIN SELECT
    customerNumber,
    salesRepEmployeeNumber
  FROM
    customers
 WHERE
    salesRepEmployeeNumber = 1370 OR
    salesRepEmployeeNumber IS NULL;
```

执行上面查询语句，得到以下结果 -

```
mysql> EXPLAIN SELECT
    customerNumber,
    salesRepEmployeeNumber
  FROM
    customers
 WHERE
    salesRepEmployeeNumber = 1370 OR
    salesRepEmployeeNumber IS NULL;
+-----+-----+-----+-----+
| id  | select_type | table      | partitions | type       | poss
ible_keys          | key           | key_len   | ref        |
rows | filtered   | Extra
+-----+-----+-----+-----+
| 1  | SIMPLE      | customers | NULL       | ref_or_null | sale
sRepEmployeeNumber | salesRepEmployeeNumber | 5          | const
29  |             100 | Using where; Using index |
+-----+-----+-----+-----+
1 row in set
```

在这个例子中，当应用优化时，`EXPLAIN` 会显示 `ref_or_null`。

如果您有一个列的组合键，MySQL 可以对任何关键部分执行优化。假设在表 `t1` 的列 `c1` 和 `c2` 上有一个索引，以下查询被优化：

```
SELECT
*
FROM
t1
WHERE
c1 IS NULL;
```

在本教程中，您已经学习了如何使用 MySQL `IS NULL` 运算符来测试值是否为 `NULL`。

在本教程中，您将学习如何使用MySQL `ORDER BY` 子句来排序结果集。

1. MySQL ORDER BY子句简介

当使用[SELECT语句](#)查询表中的数据时，结果集不按任何顺序进行排序。要对结果集进行排序，请使用 `ORDER BY` 子句。`ORDER BY` 子句允许：

- 对单个列或多个列排序结果集。
- 按升序或降序对不同列的结果集进行排序。

下面说明了 `ORDER BY` 子句的语法：

```
SELECT column1, column2, ...
FROM tb1
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...
```

`ASC` 表示升序，`DESC` 表示降序。默认情况下，如果不明确指定 `ASC` 或 `DESC`，`ORDER BY` 子句会按照升序对结果集进行排序。

下面我们来学习和练习一些使用`ORDER BY`子句的例子。

2. MySQL ORDER BY示例

请参见[示例数据库\(yiibaidb\)](#)中的 `customers` 表，`customers` 表的结构如下所示 -

```
mysql> desc customers;
+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default |
| Extra          |            |       |      |          |
+-----+-----+-----+-----+
| customerNumber | int(11)    | NO   | PRI | NULL    |
| customerName   | varchar(50) | NO   |      | NULL    |
| contactLastName | varchar(50) | NO   |      | NULL    |
| contactFirstName | varchar(50) | NO   |      | NULL    |
| phone          | varchar(50) | NO   |      | NULL    |
| addressLine1   | varchar(50) | NO   |      | NULL    |
| addressLine2   | varchar(50) | YES  |      | NULL    |
| city           | varchar(50) | NO   |      | NULL    |
| state          | varchar(50) | YES  |      | NULL    |
| postalCode     | varchar(15) | YES  |      | NULL    |
| country         | varchar(50) | NO   |      | NULL    |
| salesRepEmployeeNumber | int(11)    | YES  | MUL | NULL    |
| creditLimit    | decimal(10,2) | YES  |      | NULL    |
+-----+-----+-----+-----+
13 rows in set
```

以下查询从 `customers` 表中查询联系人，并按 `contactLastname` 升序对联系人进行排序。

```

SELECT
    contactLastname,
    contactFirstname
FROM
    customers
ORDER BY
    contactLastname;

```

执行上面查询，得到以下结果 -

```

mysql> SELECT
    contactLastname,
    contactFirstname
FROM
    customers
ORDER BY
    contactLastname;
+-----+-----+
| contactLastname | contactFirstname |
+-----+-----+
| Accorti         | Paolo          |
| Altagar, G M   | Raanan         |
| Andersen        | Mel            |
| Anton           | Carmen          |
| Ashworth        | Rachel          |
| Barajas          | Miguel          |
| Benitez          | Violeta         |
| Bennett          | Helen           |
| Berglund         | Christina       |
| Bergulfson      | Jonas           |
| Bertrand         | Marie           |
| ...             | ...             |
| Young            | Julie           |
| Young            | Mary            |
| Young            | Dorothy          |
+-----+-----+
122 rows in set

```

4.1 ORDER BY 子句

如果要按姓氏降序对联系人进行排序，请在 `ORDER BY` 子句中的 `contactLastname` 列后面指定 `DESC`，如下查询：

```
SELECT  
    contactLastname,  
    contactFirstname  
FROM  
    customers  
ORDER BY  
    contactLastname DESC;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT  
    contactLastname,  
    contactFirstname  
FROM  
    customers  
ORDER BY  
    contactLastname DESC;  
+-----+-----+  
| contactLastname | contactFirstname |  
+-----+-----+  
| Young          | Jeff           |  
| Young          | Julie          |  
| Young          | Mary           |  
| ...            | ...            |  
| Anton          | Carmen         |  
| Andersen       | Mel            |  
| Altagar, G M   | Raanan         |  
| Accorti        | Paolo          |  
+-----+-----+  
122 rows in set
```

如果要按姓氏按降序和名字按升序排序联系人，请在相应列中分别指定 `DESC` 和 `ASC`，如下所示：

```
SELECT  
    contactLastname,  
    contactFirstname  
FROM  
    customers  
ORDER BY  
    contactLastname DESC,  
    contactFirstname ASC;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT
    contactLastname,
    contactFirstname
  FROM
    customers
 ORDER BY
    contactLastname DESC,
    contactFirstname ASC;
+-----+-----+
| contactLastname | contactFirstname |
+-----+-----+
| Young          | Dorothy        |
| Young          | Jeff            |
| Young          | Julie           |
| Young          | Mary            |
| Yoshido        | Juri            |
| Walker         | Brydey          |
| Victorino      | Wendy           |
| Urs             | Braun           |
| Tseng           | Jerry           |
| ...             | ...             |
| Brown           | Julie           |
| Brown           | William          |
| Bertrand        | Marie           |
| Bergulfsen     | Jonas            |
| Berglund        | Christina       |
| Bennett         | Helen            |
| Benitez         | Violeta          |
| Barajas         | Miguel           |
| Ashworth        | Rachel           |
| Anton           | Carmen           |
| Andersen        | Mel              |
| Altagar, G M   | Raanan          |
| Accorti         | Paolo            |
+-----+-----+
122 rows in set
```

在上面的查询中， ORDER BY 子句首先按照 contactLastname 列降序对结果集进行排序，然后按照 contactFirstname 列升序对排序结果集进行排序，以生成最终结果集。

MySQL ORDER BY按表达式排序示例

ORDER BY 子句还允许您根据表达式对结果集进行排序。请参阅以下 orderdetails 表结构 -

```
mysql> desc orderdetails;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| orderNumber | int(11) | NO | PRI | NULL |
| productCode | varchar(15) | NO | PRI | NULL |
| quantityOrdered | int(11) | NO | | NULL |
| priceEach | decimal(10,2) | NO | | NULL |
| orderLineNumber | smallint(6) | NO | | NULL |
+-----+-----+-----+-----+-----+
--+
5 rows in set
```

以下查询从 orderdetails 表中选择订单行记录项目。它计算每个订单项的小计，并根据订单编号，订单行号(orderLineNumber)和小计(quantityOrdered * priceEach)对结果集进行排序。

```

SELECT
    ordernumber,
    orderlinenumber,
    quantityOrdered * priceEach
FROM
    orderdetails
ORDER BY
    ordernumber,
    orderLineNumber,
    quantityOrdered * priceEach;

```

执行上面语句，总共有 2996 行结果集，以下是部分结果集片断 -

	ordernumber	orderlinenumber	FORMAT(quantityOrdered * priceEach, 2)
▶	10100	1	1,729.21
	10100	2	2,754.50
	10100	3	4,080.00
	10100	4	1,660.12
	10101	1	4,343.56
	10101	2	2,040.10
	10101	3	1,463.85
	10101	4	2,701.50

为了使查询更易于阅读，可以按列别名进行排序，方法如下：

```

SELECT
    ordernumber,
    orderlinenumber,
    quantityOrdered * priceEach AS subtotal
FROM
    orderdetails
ORDER BY
    ordernumber,
    orderLineNumber,
    subtotal;

```

执行上面语句，总共有 2996 行结果集，以下是部分结果集片断

	ordernumber	orderlinenumber	subtotal
▶	10100	1	1729.21
	10100	2	2754.50
	10100	3	4080.00
	10100	4	1660.12
	10101	1	4343.56
	10101	2	2040.10
	10101	3	1463.85
	10101	4	2701.50
	10102	1	1768.33

上面表达式中，使用 `subtotal` 作为表达式 `quantityOrdered * priceEach` 的列别名，并根据小计别名(`subtotal`)对结果集进行排序。

MySQL ORDER BY与自定义排序顺序

`ORDER BY` 子句允许使用 `FIELD()` 函数为列中的值定义自己的自定义排序顺序。

看看下面 `orders` 表的结构如下所示 -

```
mysql> desc orders;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| orderNumber | int(11) | NO | PRI | NULL |
| orderDate | date | NO | | NULL |
| requiredDate | date | NO | | NULL |
| shippedDate | date | YES | | NULL |
| status | varchar(15) | NO | | NULL |
| comments | text | YES | | NULL |
| customerNumber | int(11) | NO | MUL | NULL |
+-----+-----+-----+-----+-----+
7 rows in set
```

例如，如果要按以下顺序基于以下状态的值对订单进行排序：

- In Process
- On Hold
- Cancelled
- Resolved

- Disputed
- Shipped

可以使用 `FIELD()` 函数将这些值映射到数值列表，并使用数字进行排序；请参阅以下查询：

```
SELECT
    orderNumber, status
FROM
    orders
ORDER BY FIELD(status,
    'In Process',
    'On Hold',
    'Cancelled',
    'Resolved',
    'Disputed',
    'Shipped');
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
    orderNumber, status
  FROM
    orders
 ORDER BY FIELD(status,
    'In Process',
    'On Hold',
    'Cancelled',
    'Resolved',
    'Disputed',
    'Shipped');
```

orderNumber	status
10420	In Process
10421	In Process
10422	In Process
10423	In Process
10424	In Process
10425	In Process
10334	On Hold
10401	On Hold
10407	On Hold
10414	On Hold
10167	Cancelled
10179	Cancelled
10248	Cancelled
10253	Cancelled
10260	Cancelled
10262	Cancelled
10164	Resolved
10327	Resolved
...	...
10413	Shipped
10416	Shipped
10418	Shipped
10419	Shipped

```
326 rows in set
```

4.1 ORDER BY 子句

在本教程中，我们使用了各种示例演示了如何使用MySQL ORDER BY 子句对结果集进行排序。

在本教程中，您将使用 `ORDER BY` 子句了解MySQL中的各种自然排序技术。

下面让我们使用一个示例数据来开始学习自然排序技术。

假设我们有一个 `items` 的表，其中包含两列：`id` 和 `item_no`。使用以下 `CREATE TABLE`语句创建 `items` 表，如下：

```
CREATE TABLE IF NOT EXISTS items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    item_no VARCHAR(255) NOT NULL
);
```

我们使用`INSERT`语句将一些数据插入到 `items` 表中：

```
INSERT INTO items(item_no)
VALUES ('1'),
       ('1C'),
       ('10Z'),
       ('2A'),
       ('2'),
       ('3C'),
       ('20D');
```

当我们查询选择数据并按 `item_no` 排序时，得到以下结果：

```
SELECT
    item_no
FROM
    items
ORDER BY item_no;
```

item_no
1
10Z
1C
2
20D
2A
3C

7 rows in set

这不是我们的预期的结果，我们想要看到的结果如下：

item_no
1
1C
2
2A
3C
10Z
20D

这被称为自然排序。不幸的是，MySQL不提供任何内置的自然排序语法或函数。
ORDER BY子句以线性方式排序字符串，即从第一个字符开始的每个字符一次。

为了克服这个问题，首先我们将 `item_no` 列分成两列：`prefix` 和 `suffix`。
`prefix` 列存储 `item_no` 的数字部分，`suffix` 列存储字母部分。然后根据这些列对数据进行排序，如下所示：

```
SELECT  
    CONCAT(prefix, suffix)  
FROM  
    items  
ORDER BY prefix, suffix;
```

查询首先对数据进行数字排序，并按字母顺序对数据进行排序。我们就得到预期的结果。

这个解决方案的缺点是必须在插入或更新之前将 `item_no` 值分成两部分。此外，当查询数据时，必须将这两列组合成一列。

如果 `item_no` 数据格式相当标准，则可以使用以下查询执行自然排序，而无需更改表的结构。

```
SELECT  
    item_no  
FROM  
    items  
ORDER BY CAST(item_no AS UNSIGNED), item_no;
```

执行上面查询语句，得到以下结果 -

```
SELECT  
    item_no  
FROM  
    items  
ORDER BY CAST(item_no AS UNSIGNED), item_no;
```

在这个查询中，首先使用 [类型转换](#) 将 `item_no` 数据转换为无符号整数。其次，使用 `ORDER BY` 子句对数字进行数字排序，然后按字母顺序排列。

下面来看看经常要处理的另一个常见的数据。

```
TRUNCATE TABLE items;

INSERT INTO items(item_no)
VALUES('A-1'),
      ('A-2'),
      ('A-3'),
      ('A-4'),
      ('A-5'),
      ('A-10'),
      ('A-11'),
      ('A-20'),
      ('A-30');
```

排序后的预期结果如下：



为了得到上面这个结果，可以使用 LENGTH 函数。请注意，LENGTH 函数返回字符串的长度。这个做法是首先对 item_no 数据进行排序，然后按列值排序，如以下查询：

```
SELECT
    item_no
FROM
    items
ORDER BY LENGTH(item_no) , item_no;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT
    item_no
FROM
    items
ORDER BY LENGTH(item_no) , item_no;
+-----+
| item_no |
+-----+
| A 1     |
| A 2     |
| A 3     |
| A 4     |
| A 5     |
| A 10    |
| A 11    |
| A 20    |
| A 30    |
+-----+
9 rows in set
```

如下所看到数据就是自然排序的。

但是，如果所有上述解决方案都不适合。则需要在应用程序层执行自然排序。一些语言支持自然排序功能，例如，[PHP](#)提供了使用自然排序算法对数组进行排序的[natsort\(\)函数](#)。

在本教程中，我们已经演示了如何在MySQL中执行自然排序的各种技术。

在本教程中，您将学习如何使用MySQL别名来提高查询的可读性。

MySQL支持两种别名，称为列别名和表别名。下面来详细看看和学习MySQL中的别名。

MySQL列的别名

有时，列的名称是一些表达式，使查询的输出很难理解。要给列一个描述性名称，可以使用列别名。

以下语句说明了如何使用列别名：

```
SELECT  
  [column_1 | expression] AS descriptive_name  
FROM table_name;
```

要给列添加别名，可以使用 `AS` 关键词后跟别名。如果别名包含空格，则必须引用以下内容：

```
SELECT  
  [column_1 | expression] AS `descriptive name`  
FROM table_name;
```

因为 `AS` 关键字是可选的，可以在语句中省略它。请注意，还可以在表达式上使用别名。

我们来看看示例数据库(yiibaidb)中的 `employees` 表，其表结构如下所示 -

```
mysql> desc employees;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| employeeNumber | int(11) | NO | PRI | NULL |
| lastName | varchar(50) | NO | | NULL |
| firstName | varchar(50) | NO | | NULL |
| extension | varchar(10) | NO | | NULL |
| email | varchar(100) | NO | | NULL |
| officeCode | varchar(10) | NO | MUL | NULL |
| reportsTo | int(11) | YES | MUL | NULL |
| jobTitle | varchar(50) | NO | | NULL |
+-----+-----+-----+-----+-----+
8 rows in set
```

以下查询选择员工的名字和姓氏，并将其组合起来生成全名。`CONCAT_WS` 函数用于连接名字和姓氏。

```
SELECT
    CONCAT_WS(' ', lastName, firstname)
FROM
    employees;
```

执行上面代码，得到以下结果 -

```
mysql> SELECT
    CONCAT_WS(' ', ' ', lastName, firstname)
FROM
    employees;
+-----+
| CONCAT_WS(' ', ' ', lastName, firstname) |
+-----+
| Murphy, Diane
| Patterson, Mary
| Firrelli, Jeff
| Patterson, William
| Bondur, Gerard
| Bow, Anthony
| Jennings, Leslie
| Thompson, Leslie
| Firrelli, Julie
| Patterson, Steve
| Tseng, Foon Yue
| Vanauf, George
| Bondur, Loui
| Hernandez, Gerard
| Castillo, Pamela
| Bott, Larry
| Jones, Barry
| Fixter, Andy
| Marsh, Peter
| King, Tom
| Nishi, Mami
| Kato, Yoshimi
| Gerard, Martin
+-----+
23 rows in set
```

在上面示例中，列标题很难阅读理解。可以为输出的标题分配一个有意义的列别名，以使其更可读，如以下查询：

```

SELECT
CONCAT_WS(' ', ' ', lastName, firstname) AS `Full name`
FROM
employees;

```

执行上面代码，得到以下结果 -

```

mysql> SELECT
CONCAT_WS(' ', ' ', lastName, firstname) AS `Full name`
FROM
employees;
+-----+
| Full name |
+-----+
| Murphy, Diane
| Patterson, Mary
| Firrelli, Jeff
...
| King, Tom
| Nishi, Mami
| Kato, Yoshimi
| Gerard, Martin
+-----+
23 rows in set

```

在MySQL中，可以使用ORDER BY，GROUP BY和HAVING子句中的列别名来引用该列。

以下查询使用 ORDER BY 子句中的列别名按字母顺序排列员工的全名：

```

SELECT
CONCAT_WS(' ', ' ', lastName, firstname) `Full name`
FROM
employees
ORDER BY
`Full name`;

```

执行上面代码，得到以下结果 -

```
mysql> SELECT
    CONCAT_WS(' ', lastName, firstname) `Full name`
  FROM
    employees
 ORDER BY
  `Full name`;
+-----+
| Full name |
+-----+
| Bondur Gerard
| Bondur Loui
| Bott Larry
| Bow Anthony
| Castillo Pamela
| Firrelli Jeff
| Firrelli Julie
| Fixter Andy
| Gerard Martin
| Hernandez Gerard
| Jennings Leslie
| Jones Barry
| Kato Yoshimi
| King Tom
| Marsh Peter
| Murphy Diane
| Nishi Mami
| Patterson Mary
| Patterson Steve
| Patterson William
| Thompson Leslie
| Tseng Foon Yue
| Vanauf George
+-----+
23 rows in set
```

以下语句查询总金额大于 60000 的订单。它在 GROUP BY 和 HAVING 子句中使用列别名。

```

SELECT
    orderNumber `Order no.`,
    SUM(priceEach * quantityOrdered) total
FROM
    orderdetails
GROUP BY
    `Order no.`
HAVING
    total > 60000;

```

执行上面查询语句，得到以下结果 -

```

mysql> SELECT
    orderNumber `Order no.`,
    SUM(priceEach * quantityOrdered) total
FROM
    orderdetails
GROUP BY
    `Order no.`
HAVING
    total > 60000;
+-----+-----+
| Order no. | total |
+-----+-----+
| 10165 | 67392.85 |
| 10287 | 61402.00 |
| 10310 | 61234.67 |
+-----+-----+
3 rows in set

```

请注意，不能在 WHERE 子句中使用列别名。原因是当 MySQL 评估求值 WHERE 子句时， SELECT 子句中指定的列的值可能尚未确定。

MySQL 表的别名

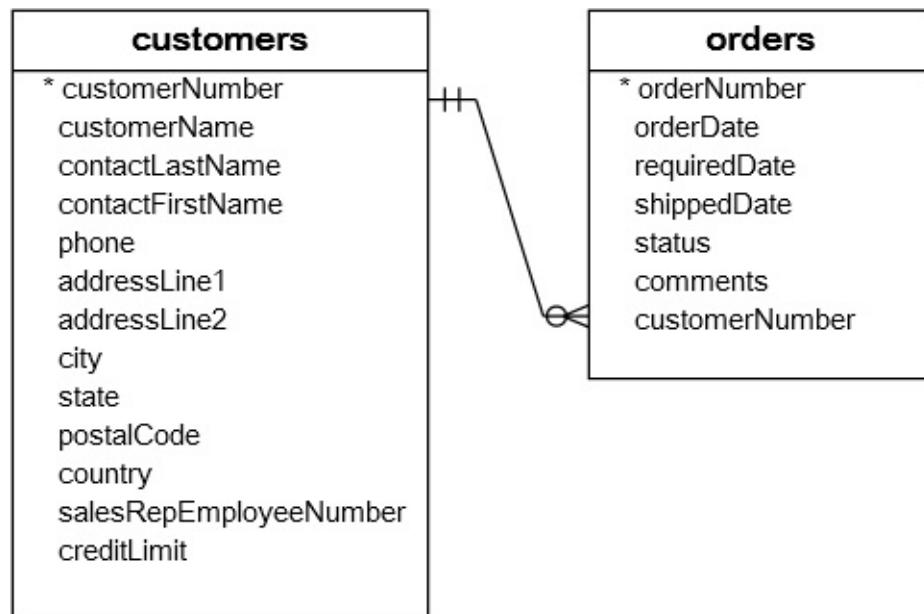
可以使用别名为表添加不同的名称。使用 AS 关键字在表名称分配别名，如下查询语句语法：

```
table_name AS table_alias
```

该表的别名称为表别名。像列别名一样，`AS` 关键字是可选的，所以完全可以省略它。

一般在包含 `INNER JOIN`，`LEFT JOIN`，`self join` 子句和子查询的语句中使用表别名。

下面来看看客户(`customers`)和订单(`orders`)表，它们的ER图如下所示 -



两个表都具有相同的列名称：`customerNumber`。如果不使用表别名来指定是哪个表中的 `customerNumber` 列，则执行查询时将收到类似以下错误消息：

```
Error Code: 1052. Column 'customerNumber' in on clause is ambiguous
```

为避免此错误，应该使用表别名来限定 `customerNumber` 列：

```

SELECT
    customerName,
    COUNT(o.orderNumber) total
FROM
    customers c
INNER JOIN orders o ON c.customerNumber = o.customerNumber
GROUP BY
    customerName
HAVING total >=5
ORDER BY
    total DESC;

```

执行上面查询语句，得到以下结果 -

```

mysql> SELECT
    customerName,
    COUNT(o.orderNumber) total
FROM
    customers c
INNER JOIN orders o ON c.customerNumber = o.customerNumber
GROUP BY
    customerName
HAVING total >=5
ORDER BY
    total DESC;
+-----+-----+
| customerName | total |
+-----+-----+
| Euro- Shopping Channel | 26 |
| Mini Gifts Distributors Ltd. | 17 |
| Reims Collectables | 5 |
| Down Under Souveniers, Inc | 5 |
| Danish Wholesale Imports | 5 |
| Australian Collectors, Co. | 5 |
| Dragon Souveniers, Ltd. | 5 |
+-----+-----+
7 rows in set

```

上面的查询从客户(`customers`)和订单(`orders`)表中选择客户名称和订单数量。它使用 `c` 作为 `customers` 表的表别名，`o` 作为 `orders` 表的表别名。`customers` 和 `orders` 表中的列通过表别名(`c` 和 `o`)引用。

如果您不在上述查询中使用别名，则必须使用表名称来引用其列，这样的会使得查询冗长且可读性较低，如下 -

```
SELECT
    customers.customerName,
    COUNT(orders.orderNumber) total
FROM
    customers
INNER JOIN orders ON customers.customerNumber = orders.customerN
umber
GROUP BY
    customerName
ORDER BY
    total DESC
```

在本教程中，我们向演示了如何使用MySQL别名，使查询更易于阅读和更易于理解。

在本教程中，您将学习如何使用MySQL `INNER JOIN` 子句根据连接条件从多个表中查询选择数据。

MySQL INNER JOIN子句介绍

MySQL `INNER JOIN` 子句将一个表中的行与其他表中的行进行匹配，并允许从两个表中查询包含列的行记录。

`INNER JOIN` 子句是 `SELECT` 语句的可选部分，它出现在 `FROM` 子句之后。

在使用 `INNER JOIN` 子句之前，必须指定以下条件：

- 首先，在 `FROM` 子句中指定主表。
- 其次，表中要连接的主表应该出现在 `INNER JOIN` 子句中。理论上说，可以连接多个其他表。但是，为了获得更好的性能，应该限制要连接的表的数量（最好不要超过三个表）。
- 第三，连接条件或连接谓词。连接条件出现在 `INNER JOIN` 子句的 `ON` 关键字之后。连接条件是将主表中的行与其他表中的行进行匹配的规则。

`INNER JOIN` 子句的语法如下：

```
SELECT column_list
FROM t1
INNER JOIN t2 ON join_condition1
INNER JOIN t3 ON join_condition2
...
WHERE where_conditions;
```

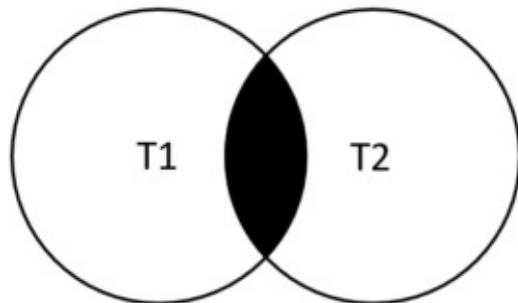
假设使用 `INNER JOIN` 子句连接两个表：`t1` 和 `t2`，我们来简化上面的语法。

```
SELECT column_list
FROM t1
INNER JOIN t2 ON join_condition;
```

对于 `t1` 表中的每一行，`INNER JOIN` 子句将它与 `t2` 表的每一行进行比较，以检查它们是否都满足连接条件。当满足连接条件时，`INNER JOIN` 将返回由 `t1` 和 `t2` 表中的列组成的新行。

请注意，`t1` 和 `t2` 表中的行必须根据连接条件进行匹配。如果找不到匹配项，查询将返回一个空结果集。当连接超过 2 个表时，也应用此逻辑。

以下维恩图说明了 `INNER JOIN` 子句的工作原理。结果集中的行必须出现在两个表中：`t1` 和 `t2`，如两个圆的交叉部分所示 -



在 MySQL INNER JOIN 中避免列错误

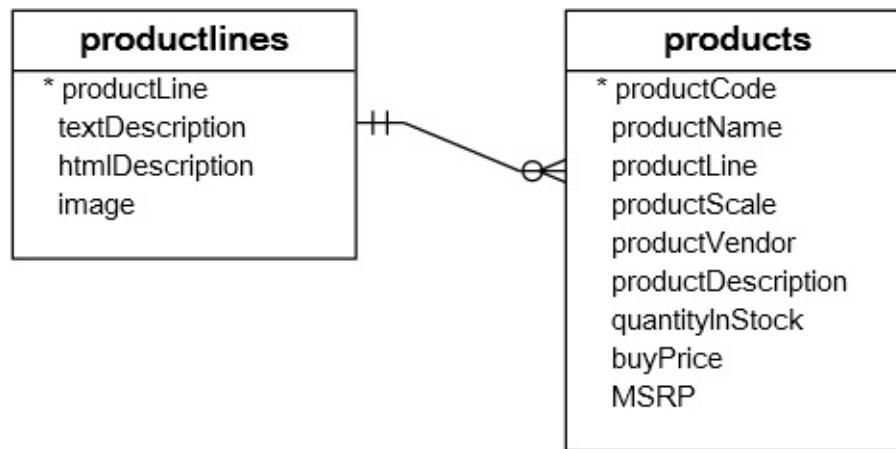
如果连接具有相同列名称的多个表，则必须使用表限定符引用 `SELECT` 和 `ON` 子句的列，以避免列错误。

例如，如果 `t1` 和 `t2` 表都具有名为 `c` 的一个相同列名，则必须在 `SELECT` 和 `ON` 子句中使用表限定符，如使用 `t1.c` 或 `t2.c` 指定引用是那个表中的 `c` 列。

为了节省书写表限定符的时间，可以在查询中使用表别名。例如，可以长名称 `verylonglonglong_tablelename` 表使用表别名，并使用 `t.column` 引用其列，而不是使用 `verylonglonglong_tablelename.column`，但是如果喜欢书写或使用这么长的表名称，那么也应该允许你的开发伙伴骂你几句类似：傻逼~等这样的话！

MySQL INNER JOIN示例

下面来看看示例数据库([yiibaidb](#))中的产品(`products`)和产品线(`productlines`)表。它们的 `ER` 图如下所示 -



在上面图中，`products` 表中的 `productLine` 列参考引用 `productlines` 表的 `productline` 列。`products` 表中的 `productLine` 列称为外键列。

通常，连接具有外键关系的表，如产品线(`productlines`)和产品(`products`)表。现在，如果想获取以下数据 -

- 获取 `products` 表中的 `productCode` 和 `productName` 列的值。
- 获取 `productlines` 表产品线的描述 - `textDescription` 列的值。

为此，需要通过使用 `INNER JOIN` 子句根据 `productline` 列匹配行来从两个表中查询选择数据，如下所示：

```

SELECT
    productCode,
    productName,
    textDescription
FROM
    products t1
    INNER JOIN
    productlines t2 ON t1.productline = t2.productline;
  
```

执行上面查询，得到下面的结果(部分)-

	productCode	productName	textDescription
	S10_1949	1952 Alpine Renault 1300	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S10_4757	1972 Alfa Romeo GTA	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S10_4962	1962 LanciaA Delta 16V	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S12_1099	1968 Ford Mustang	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S12_1108	2001 Ferrari Enzo	Attention car enthusiasts: Make your wildest car ownership dreams come true.

5.2 INNER JOIN

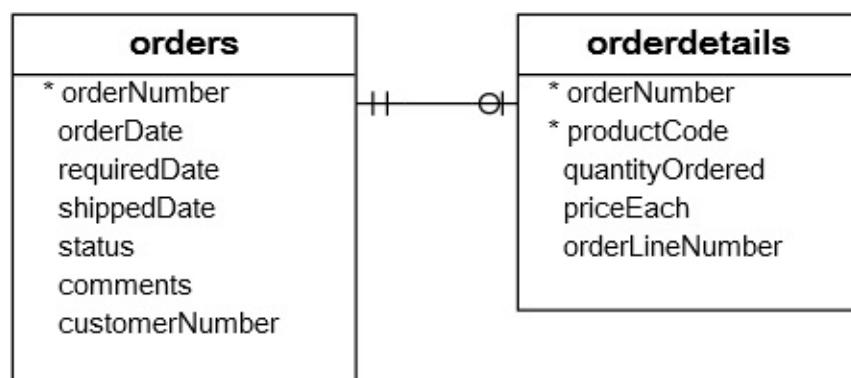
由于两个表的连接列是使用同一个列：`productline`，因此可以使用以下语法：

```
SELECT
    productCode,
    productName,
    textDescription
FROM
    products
        INNER JOIN
    productlines USING (productline);
```

上面语句返回相同的结果集，但是使用此语法，不必使用表的别名。

MySQL INNER JOIN GROUP BY子句

请参阅以下订单和订单详细表，`orders` 表和 `orderdetails` 表的结构如下所示 -



可以使用具有 `GROUP BY` 子句的 `INNER JOIN` 子句

从 `orders` 和 `orderdetails` 表中获取订单号，订单状态和总销售额，如下所示：

```
SELECT
    T1.orderNumber,
    status,
    SUM(quantityOrdered * priceEach) total
FROM
    orders AS T1
        INNER JOIN
    orderdetails AS T2 ON T1.orderNumber = T2.orderNumber
GROUP BY orderNumber;
```

5.2 INNER JOIN

执行上面查询，结果如下所示(部分) -

	orderNumber	status	total
	10100	Shipped	10223.83
	10101	Shipped	10549.01
	10102	Shipped	5494.78
	10103	Shipped	50218.95
	10104	Shipped	40206.20

类似地，以下语句查询与上述得到结果相同：

```
SELECT
    orderNumber,
    status,
    SUM(quantityOrdered * priceEach) total
FROM
    orders
        INNER JOIN
    orderdetails USING (orderNumber)
GROUP BY orderNumber;
```

MySQL INNER JOIN 使用等于以外的运算符

到目前为止，您已经看到连接谓词使用相等的运算符(=)来匹配行。但是也可以使用大于(>)，小于(<)和不等于(<>)运算符的其他运算符来形成连接谓词。

以下查询使用少于(<)连接来查找低于代码为 S10_1678 的产品的销售价格的制造商建议零售价(MSRP)的所有产品。

```
SELECT
    orderNumber,
    productName,
    msrp,
    priceEach
FROM
    products p
        INNER JOIN
    orderdetails o ON p.productcode = o.productcode
        AND p.msrp > o.priceEach
WHERE
    p.productcode = 'S10_1678';
```

执行上面查询语句，得到以下输出结果 -

```

mysql> SELECT
    orderNumber,
    productName,
    msrp,
    priceEach
  FROM
    products p
    INNER JOIN
    orderdetails o ON p.productcode = o.productcode
    AND p.msrp > o.priceEach
 WHERE
    p.productcode = 'S10_1678';
+-----+-----+-----+-----+
| orderNumber | productName | msrp | p
riceEach |
+-----+-----+-----+-----+
| 10107 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 8
1.35
| 10121 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 8
6.13
| 10134 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 9
0.92
| 10145 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 7
6.56
| 10159 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 8
1.35
| 10168 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 9
4.74
| 10399 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 7
7.52
| 10403 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 8
5.17
| ...
| 10417 | 1969 Harley Davidson Ultimate Chopper | 95.7 | 7
9.43
+-----+-----+-----+-----+
| 26 rows in set

```

5.2 INNER JOIN

在本教程中，您已经学会了如何使用MySQL `INNER JOIN` 来查询来自多个表中的数据。

在本教程中，您将了解MySQL `LEFT JOIN` 子句以及如何将其应用于从两个或多个数据库表查询数据。

1. MySQL LEFT JOIN简介

MySQL `LEFT JOIN` 子句允许您从两个或多个数据库表查询数据。`LEFT JOIN` 子句是`SELECT`语句的可选部分，出现在 `FROM` 子句之后。

我们假设要从两个表 `t1` 和 `t2` 查询数据。以下语句说明了连接两个表的 `LEFT JOIN` 子句的语法：

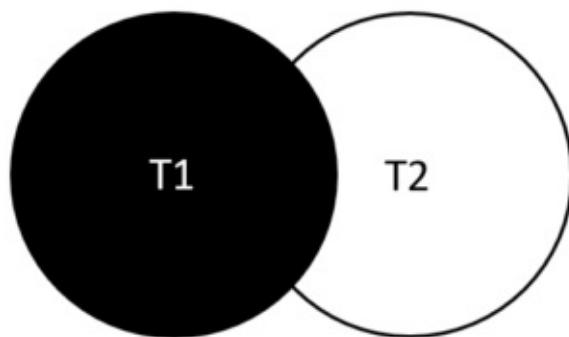
```
SELECT
    t1.c1, t1.c2, t2.c1, t2.c2
FROM
    t1
        LEFT JOIN
    t2 ON t1.c1 = t2.c1;
```

当使用 `LEFT JOIN` 子句将 `t1` 表加入 `t2` 表时，如果来自左表 `t1` 的行与基于连接条件(`t1.c1 = t2.c1`)的右表 `t2` 匹配，则该行将被包含在结果集中。

如果左表中的行与右表中的行不匹配，则还将选择左表中的行并与右表中的“假”行组合。“假”行对于 `SELECT` 子句中的所有相应列都包含 `NULL` 值。

换句话说，`LEFT JOIN` 子句允许您从匹配的左右表中查询选择行记录，连接左表(`t1`)中的所有行，即使在右表(`t2`)中找不到匹配的行也显示出来，但使用 `NULL` 值代替。

下图可帮助您可视化 `LEFT JOIN` 子句的工作原理。两个圆圈之间的交点是两个表中匹配的行，左圆的剩余部分(白色部分)是 `t1` 表中不存在 `t2` 表中任何匹配行的行。因此，左表中的所有行都包含在结果集中。

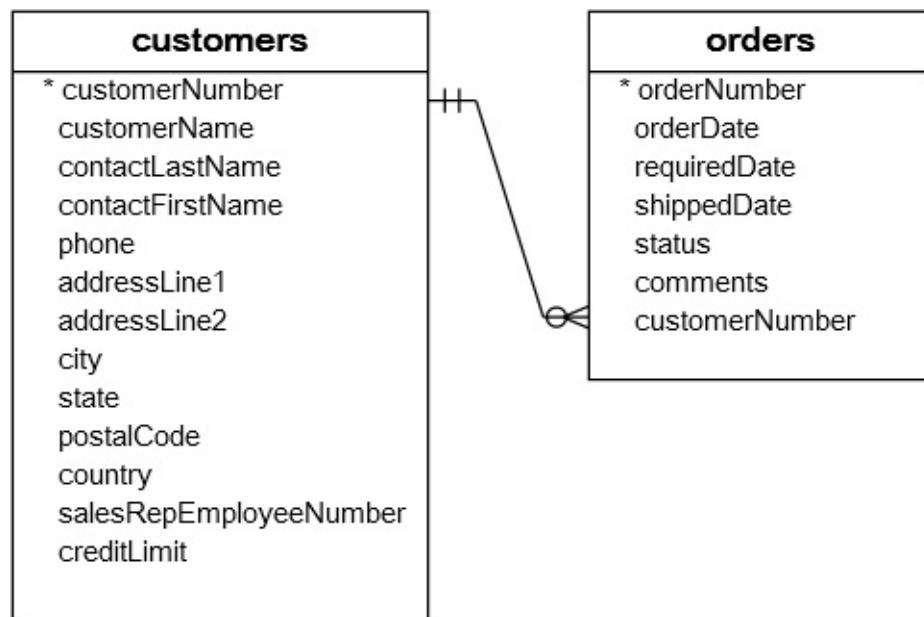


请注意，如果这些子句在查询中可用，返回的行也必须与 **WHERE** 和 **HAVING** 子句中的条件相匹配。

2. MySQL LEFT JOIN示例

2.1 使用MySQL LEFT JOIN子句来连接两个表

我们来看看在 [示例数据库\(yiibaidb\)](#) 中的两个表：订单表和客户表，两个表的 *ER* 图如下所示 -



在上面的数据库图中：

- 订单(`orders`)表中的每个订单必须属于客户(`customers`)表中的客户。
- 客户(`customers`)表中的每个客户在订单(`orders`)表中可以有零个或多个订单。

要查询每个客户的所有订单，可以使用 `LEFT JOIN` 子句，如下所示：

```

SELECT
    c.customerNumber,
    c.customerName,
    orderNumber,
    o.status
FROM
    customers c
LEFT JOIN orders o ON c.customerNumber = o.customerNumber;
  
```

5.3 LEFT JOIN

执行上面查询语句，得到以下结果(部分) -

```
mysql> SELECT
    c.customerNumber,
    c.customerName,
    orderNumber,
    o.status
  FROM
    customers c
  LEFT JOIN orders o ON c.customerNumber = o.customerNumber;
+-----+-----+-----+
| customerNumber | customerName | orderNum
ber | status |
+-----+-----+-----+
| 103 | Atelier graphique | 10
| 123 | Shipped |
| 103 | Atelier graphique | 10
| 298 | Shipped |
省略部分 ...
| 477 | Mit Vergngen & Co. | NULL
| NULL | Kremlin Collectables, Co. | NULL
| 481 | Raanan Stores, Inc | NULL
| NULL | Iberia Gift Imports, Corp. | 10
| 184 | Shipped |
| 484 | Iberia Gift Imports, Corp. | 10
| 303 | Shipped |
| 486 | Motor Mint Distributors Inc. | 10
| 109 | Shipped |
| 486 | Motor Mint Distributors Inc. | 10
| 236 | Shipped |
+-----+-----+-----+
350 rows in set
```

左表是 `customers` 表，因此，所有客户都包含在结果集中。但是，结果集中有一些行具有客户数据，但没有订单数据。如：`customerNumber` 列值为：`477`，`480` 等。这些行中的订单数据为 `NULL`。也就是说这些客户在 `orders` 表中没有任何订单(未购买过任何产品)。

因为我们使用相同的列名(`orderNumber`)来连接两个表，所以可以使用以下语法使查询更短：

```
SELECT
    c.customerNumber,
    customerName,
    orderNumber,
    status
FROM
    customers c
LEFT JOIN orders USING (customerNumber);
```

在上面查询语句中，下面的子句 -

```
USING (customerNumber)
```

相当于 -

```
ON c.customerNumber = o.customerNumber
```

如果使用 `INNER JOIN` 子句替换 `LEFT JOIN` 子句，则只能获得至少有下过一个订单的客户。

2.2 使用 MySQL LEFT JOIN 子句来查找不匹配的行

当您想要找到右表中与不匹配的左表中的行时，`LEFT JOIN` 子句非常有用。要查询两个表之间的不匹配行，可以向 `SELECT` 语句添加一个 `WHERE` 子句，以仅查询右表中的列值包含 `NULL` 值的行。

例如，要查找没有下过订单的所有客户，请使用以下查询：

```

SELECT
    c.customerNumber,
    c.customerName,
    orderNumber,
    o.status
FROM
    customers c
        LEFT JOIN
    orders o ON c.customerNumber = o.customerNumber
WHERE
    orderNumber IS NULL;

```

执行上面查询语句，得到以下结果 -

```

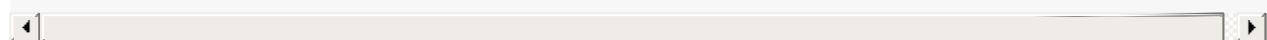
mysql> SELECT
    c.customerNumber,
    c.customerName,
    orderNumber,
    o.status
FROM
    customers c
        LEFT JOIN
    orders o ON c.customerNumber = o.customerNumber
WHERE
    orderNumber IS NULL;
+-----+-----+-----+
+-----+-----+
| customerNumber | customerName | orderNumber |
| status         |             |             |
+-----+-----+-----+
+-----+-----+
|      125 | Havel & Zbyszek Co | NULL       |
|     NULL |             |             |
|      168 | American Souvenirs Inc | NULL       |
|     NULL |             |             |
|      169 | Porto Imports Co. | NULL       |
|     NULL |             |             |
|      206 | Asian Shopping Network, Co | NULL       |
|     NULL |             |             |
|      223 | Natrlich Autos | NULL       |
|             |             |             |
+-----+-----+-----+

```

5.3 LEFT JOIN

NULL			
	237	ANG Resellers	NULL
NULL	247	Messner Shopping Network	NULL
NULL	273	Franken Gifts, Co	NULL
NULL	293	BG&E Collectables	NULL
NULL	303	Schuyler Imports	NULL
NULL	307	Der Hund Imports	NULL
NULL	335	Cramer Speziallitten, Ltd	NULL
NULL	348	Asian Treasures, Inc.	NULL
NULL	356	SAR Distributors, Co	NULL
NULL	361	Kommission Auto	NULL
NULL	369	Lisboa Souveniers, Inc	NULL
NULL	376	Precious Collectables	NULL
NULL	409	Stuttgart Collectable Exchange	NULL
NULL	443	Feuer Online Stores, Inc	NULL
NULL	459	Warburg Exchange	NULL
NULL	465	Anton Designs, Ltd.	NULL
NULL	477	Mit Vergngen & Co.	NULL
NULL	480	Kremlin Collectables, Co.	NULL
NULL	481	Raanan Stores, Inc	NULL
NULL			

24 rows in set



3. WHERE子句与ON子句中的条件

请参见以下示例。

```
SELECT
    o.orderNumber,
    customerNumber,
    productCode
FROM
    orders o
    LEFT JOIN
    orderDetails USING (orderNumber)
WHERE
    orderNumber = 10123;
```

在本示例中，我们使用 `LEFT JOIN` 子句来查询 `orders` 表和 `orderDetails` 表中的数据。该查询返回订单号为 `10123` 的订单及其购买产品明细信息(如果有的话)。

```

mysql> SELECT
    o.orderNumber,
    customerNumber,
    productCode
  FROM
    orders o
    LEFT JOIN
    orderDetails USING (orderNumber)
 WHERE
    orderNumber = 10123;
+-----+-----+-----+
| orderNumber | customerNumber | productCode |
+-----+-----+-----+
| 10123      | 103          | S18_1589   |
| 10123      | 103          | S18_2870   |
| 10123      | 103          | S18_3685   |
| 10123      | 103          | S24_1628   |
+-----+-----+-----+
4 rows in set

```

但是，如果将条件从 WHERE 子句移动到 ON 子句：

```

SELECT
    o.orderNumber,
    customerNumber,
    productCode
  FROM
    orders o
    LEFT JOIN
    orderDetails d ON o.orderNumber = d.orderNumber
    AND o.orderNumber = 10123;

```

想想上面代码将会输出什么结果 -

```
mysql> SELECT
    o.orderNumber,
    customerNumber,
    productCode
  FROM
    orders o
    LEFT JOIN
    orderDetails d ON o.orderNumber = d.orderNumber
    AND o.orderNumber = 10123;
+-----+-----+-----+
| orderNumber | customerNumber | productCode |
+-----+-----+-----+
| 10123      | 103          | S18_1589   |
| 10123      | 103          | S18_2870   |
| 10123      | 103          | S18_3685   |
| 10123      | 103          | S24_1628   |
| 10298      | 103          | NULL       |
| 10345      | 103          | NULL       |
| 10124      | 112          | NULL       |
| ...         | ...          | ...        |
| 10179      | 496          | NULL       |
| 10360      | 496          | NULL       |
| 10399      | 496          | NULL       |
+-----+-----+-----+
329 rows in set
```

请注意，对于**INNER JOIN**子句，**ON**子句中的条件等同于**WHERE**子句中的条件。

在本教程中，我们解释了MySQL **LEFT JOIN**子句，并向您展示了如何将使用它来从多个数据库表中查询数据。

在本教程中，您将了解如何使用连接语句将表连接到表自身，即，在同一张表上自己连接自己。

在之前的教程中，已经学习了如何使用 **INNER JOIN**，**LEFT JOIN** 或 **CROSS JOIN** 子句将表连接到其他表。但是，有一个特殊情况，需要将表自身连接，这被称为自连接。

当您想将表中行与同一表中的其他行组合时，可以使用自连接。要执行自连接操作必须使用 **表别名** 来帮助 MySQL 在单个查询中区分左表与同一张表的右表。

MySQL 自连接的例子

我们来看看示例数据库([yii2aidb](#))中的 `employees` 表，其表结构如下所示 -

要获得整个组织结构，可以使用 `employeeNumber` 和 `reportsTo` 列将 `employees` 表连接自身。`employees` 表有两个角色：一个是经理，另一个是直接报告者(即，下属员工)。

```
SELECT
    CONCAT(m.lastname, ' ', m.firstname) AS 'Manager',
    CONCAT(e.lastname, ' ', e.firstname) AS 'Direct report'
FROM
    employees e
        INNER JOIN
    employees m ON m.employeeNumber = e.reportsTo
ORDER BY manager;
```

执行上面查询，得到以下结果 -

Manager	Direct report
Bondur, Gerard	Jones, Barry
Bondur, Gerard	Castillo, Pamela
Bondur, Gerard	Bondur, Loui
Bondur, Gerard	Bott, Larry
Bondur, Gerard	Gerard, Martin
Bondur, Gerard	Hernandez, Gerard
Bow, Anthony	Vanauf, George
Bow, Anthony	Patterson, Steve
Bow, Anthony	Thompson, Leslie
Bow, Anthony	Tseng, Foon Yue
Bow, Anthony	Firrelli, Julie
Bow, Anthony	Jennings, Leslie
Murphy, Diane	Firrelli, Jeff
Murphy, Diane	Patterson, Mary
Nishi, Mami	Kato, Yoshimi
Patterson, Mary	Bow, Anthony
Patterson, Mary	Patterson, William
Patterson, Mary	Bondur, Gerard
Patterson, Mary	Nishi, Mami
Patterson, William	Marsh, Peter
Patterson, William	King, Tom
Patterson, William	Fixter, Andy

22 rows in set

在上述输出中，只能看到有经理的员工。但是，由于 `INNER JOIN` 子句，所以看不到总经理。总经理是没有任何经理的员工，或者他的经理人是 `NULL`。

我们将上述查询中的 `INNER JOIN` 子句更改为 `LEFT JOIN` 子句，以包括总经理。如果管理员名称为 `NULL`，则还需要使用 `IFNULL` 函数来显示总经理。

```

SELECT
    IFNULL(CONCAT(m.lastname, ' ', m.firstname),
           'Top Manager') AS 'Manager',
    CONCAT(e.lastname, ' ', e.firstname) AS 'Direct report'
FROM
    employees e
        LEFT JOIN
    employees m ON m.employeeNumber = e.reportsto
ORDER BY manager DESC;

```

执行上面查询，得到以下结果 -

Manager	Direct report
Top Manager	Murphy, Diane
Patterson, William	Fixter, Andy
Patterson, William	Marsh, Peter
Patterson, William	King, Tom
Patterson, Mary	Bondur, Gerard
Patterson, Mary	Nishi, Mami
Patterson, Mary	Bow, Anthony
Patterson, Mary	Patterson, William
Nishi, Mami	Kato, Yoshimi
Murphy, Diane	Patterson, Mary
Murphy, Diane	Firrelli, Jeff
Bow, Anthony	Tseng, Foon Yue
Bow, Anthony	Firrelli, Julie
Bow, Anthony	Jennings, Leslie
Bow, Anthony	Vanauf, George
Bow, Anthony	Patterson, Steve
Bow, Anthony	Thompson, Leslie
Bondur, Gerard	Bott, Larry
Bondur, Gerard	Gerard, Martin
Bondur, Gerard	Hernandez, Gerard
Bondur, Gerard	Jones, Barry
Bondur, Gerard	Castillo, Pamela
Bondur, Gerard	Bondur, Loui

23 rows in set

通过使用MySQL自连接，可以通过将 `customers` 表连接自身来显示位于同一个城市的客户列表。参考以下查询语句 -

```
SELECT
    c1.city, c1.customerName, c2.customerName
FROM
    customers c1
        INNER JOIN
    customers c2 ON c1.city = c2.city
        AND c1.customerName > c2.customerName
ORDER BY c1.city;
```

执行上面查询语句，得到以下结果 -

city	customerName	customerName
Auckland	Kelly's Gift Shop	Down Under Souveniers, Inc
Auckland	Kelly's Gift Shop	GiftsForHim.com
Auckland	GiftsForHim.com	Down Under Souveniers, Inc
Boston	Gifts4AllAges.com	Diecast Collectables
Brickhaven	Collectables For Less Inc.	Auto Moto Class
Brickhaven	Online Mini Collectables	Auto Moto Class
Brickhaven	Online Mini Collectables	Collectables For Less Inc.
Cambridge	Martas Replicas Co.	Cambridge Collectables Co.
Frankfurt	Messner Shopping Network	Blauer See Auto, Co.
Glendale	Gift Ideas Corp.	Boards & Toys Co

Lisboa rs, Inc	Porto Imports Co.	Lisboa Souvenie
London ift Stores, Ltd	Stylish Desk Decors, Co.	Double Decker G
Madrid Ltd.	Corrida Auto Replicas, Ltd	Anton Designs,
Madrid	Corrida Auto Replicas, Ltd	ANG Resellers
Madrid Ltd.	CAF Imports	Anton Designs,
Madrid	CAF Imports	ANG Resellers
Madrid Ltd.	Euro+ Shopping Channel	Anton Designs,
Madrid	Euro+ Shopping Channel	ANG Resellers
Madrid	Corrida Auto Replicas, Ltd	CAF Imports
Madrid	Euro+ Shopping Channel	CAF Imports
Madrid	Anton Designs, Ltd.	ANG Resellers
Madrid plicas, Ltd	Euro+ Shopping Channel	Corrida Auto Re
Nantes ue	La Rochelle Gifts	Atelier graphiq
New Bedford m	Mini Creations Ltd.	FunGiftIdeas.co
New Haven irs Inc	Super Scale Inc.	American Souven
NYC Inc.	Muscle Machine Inc	Classic Legends
NYC	Vitachrome Inc.	Land of Toys Inc
NYC	Vitachrome Inc.	Microscale Inc.
NYC Inc.	Land of Toys Inc.	Classic Legends
NYC	Muscle Machine Inc	Land of Toys Inc
NYC	Muscle Machine Inc	Microscale Inc.

NYC Inc	Vitachrome Inc.	Muscle Machine
NYC Inc.	Microscale Inc.	Classic Legends
NYC Inc.	Vitachrome Inc.	Classic Legends
NYC	Microscale Inc.	Land of Toys Inc
Paris dance, Co.	Lyon Souveniers	La Corne D'abon
Paris it	Lyon Souveniers	Auto Canal+ Pet
Paris it	La Corne D'abondance, Co.	Auto Canal+ Pet
Philadelphia eas, Inc	Motor Mint Distributors Inc.	Classic Gift Id
San Francisco Ideas Co.	Mini Wheels Co.	Corporate Gift
Singapore Network, Co	Dragon Souveniers, Ltd.	Asian Shopping
Singapore rs, Ltd.	Handji Gifts& Co	Dragon Souvenie
Singapore Network, Co	Handji Gifts& Co	Asian Shopping

43 rows in set		

我们通过以下连接条件连接了 `customers` 表：

- 指定 `c1.city = c2.city` 以确保两个表的客户都是来自相同的城市。
- `c.customerName > c2.customerName` 以确保不要得到相同的客户。

在本教程中，我们向您介绍了MySQL自连接，可以通过使用 `INNER JOIN` 或 `LEFT JOIN` 子句将一个表连接到自身。

在本教程中，您将了解 MySQL `CROSS JOIN` 子句以及如何应用它来解决一些有趣的数据问题。

MySQL CROSS JOIN子句简介

`CROSS JOIN` 子句从连接的表返回行的笛卡儿乘积。

假设使用 `CROSS JOIN` 连接两个表。结果集将包括两个表中的所有行，其中结果集中的每一行都是第一个表中的行与第二个表中的行的组合。当连接的表之间没有关系时，会使用这种情况。

要特别注意的是，如果每个表有 1000 行，那么结果集中就有 $1000 \times 1000 = 1,000,000$ 行，那么数据量是非常巨大的。

下面说明连接两个表：`T1` 和 `T2` 的 `CROSS JOIN` 子句的语法：

```
SELECT
  *
FROM
  T1
  CROSS JOIN
  T2;
```

请注意，与 `INNER JOIN` 或 `LEFT JOIN` 子句不同，`CROSS JOIN` 子句不具有连接条件。

如果添加了 `WHERE` 子句，如果 `T1` 和 `T2` 有关系，则 `CROSS JOIN` 的工作方式与 `INNER JOIN` 子句类似，如以下查询所示：

```
SELECT
  *
FROM
  T1
  CROSS JOIN
  T2
WHERE
  T1.id = T2.id;
```

MySQL CROSS JOIN子句示例

下面我们将使用以下几个表来演示 `CROSS JOIN` 的工作原理。

```
CREATE DATABASE IF NOT EXISTS testdb;
USE testdb;

DROP TABLE IF EXISTS products;

CREATE TABLE products (
    id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(100),
    price DECIMAL(13 , 2 )
);

DROP TABLE IF EXISTS sales;

CREATE TABLE stores (
    id INT PRIMARY KEY AUTO_INCREMENT,
    store_name VARCHAR(100)
);

DROP TABLE IF EXISTS sales;

CREATE TABLE sales (
    product_id INT,
    store_id INT,
    quantity DECIMAL(13 , 2 ) NOT NULL,
    sales_date DATE NOT NULL,
    PRIMARY KEY (product_id , store_id),
    FOREIGN KEY (product_id)
        REFERENCES products (id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (store_id)
        REFERENCES stores (id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

上面语句中，创建了三个表：

- 产品(`products`)表包含产品编号，产品名称和销售价格等产品主要数据。

- 商店(stores)表包含销售产品的商店信息。
- 销售(sales)表包含在特定商店按数量和日期销售的产品。

假设有三个产品： iPhone ， iPad 和 Macbook Pro ，在北部(North)和南部(South)的这两个商店中出售。

```
INSERT INTO products(product_name, price)
VALUES('iPhone', 699),
      ('iPad', 599),
      ('Macbook Pro', 1299);

INSERT INTO stores(store_name)
VALUES('North'),
      ('South');

INSERT INTO sales(store_id, product_id, quantity, sales_date)
VALUES(1,1,20,'2017-01-02'),
      (1,2,15,'2017-01-05'),
      (1,3,25,'2017-01-05'),
      (2,1,30,'2017-01-02'),
      (2,2,35,'2017-01-05');
```

要获得每个商店和每个产品的总销售额，您可以计算销售额，并按商店和产品分组如下：

```
SELECT
    store_name,
    product_name,
    SUM(quantity * price) AS revenue
FROM
    sales
        INNER JOIN
    products ON products.id = sales.product_id
        INNER JOIN
    stores ON stores.id = sales.store_id
GROUP BY store_name , product_name;
```

执行上面查询，得到以下结果 -

```

mysql> SELECT
    store_name,
    product_name,
    SUM(quantity * price) AS revenue
FROM
    sales
        INNER JOIN
    products ON products.id = sales.product_id
        INNER JOIN
    stores ON stores.id = sales.store_id
GROUP BY store_name , product_name;
+-----+-----+-----+
| store_name | product_name | revenue |
+-----+-----+-----+
| North     | iPad         | 8985.0000 |
| North     | iPhone        | 13980.0000 |
| North     | Macbook Pro   | 32475.0000 |
| South     | iPad          | 20965.0000 |
| South     | iPhone         | 20970.0000 |
+-----+-----+-----+
5 rows in set

```

现在，如果你想知道哪个商店中的哪些产品的没有销售怎么办？上面的查询无法回答这个问题。

要解决这个问题，可以使用 `CROSS JOIN` 子句。

首先，使用 `CROSS JOIN` 子句来获取所有商店和产品的组合：

```

SELECT
    store_name, product_name
FROM
    stores AS a
        CROSS JOIN
    products AS b;

```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
    store_name, product_name
  FROM
    stores AS a
    CROSS JOIN
    products AS b;
+-----+-----+
| store_name | product_name |
+-----+-----+
| North     | iPhone      |
| South     | iPhone      |
| North     | iPad        |
| South     | iPad        |
| North     | Macbook Pro |
| South     | Macbook Pro |
+-----+-----+
6 rows in set
```

接下来，将上述查询的结果与按商店和产品返回总销售额的查询相结合。以下查询说明了这个想法：

```

SELECT
    b.store_name,
    a.product_name,
    IFNULL(c.revenue, 0) AS revenue
FROM
    products AS a
        CROSS JOIN
    stores AS b
        LEFT JOIN
    (SELECT
        stores.id AS store_id,
        products.id AS product_id,
        store_name,
        product_name,
        ROUND(SUM(quantity * price), 0) AS revenue
    FROM
        sales
    INNER JOIN products ON products.id = sales.product_id
    INNER JOIN stores ON stores.id = sales.store_id
    GROUP BY store_name, product_name) AS c ON c.store_id = b.id
        AND c.product_id= a.id
ORDER BY b.store_name;

```

请注意，如果收入为 `NULL` (表示商店没有销售的产品)，则查询使用`IFNULL`函数返回 `0`。

通过这样使用 `CROSS JOIN` 子句，可以解决类似这样的问题，例如销售人员按月查找销售收入，即使推销员在特定月份没有销售产品。

在本教程中，您将学习如何使用MySQL `GROUP BY` 根据指定列或表达式的值将行进行分组到子组。

MySQL GROUP BY子句简介

`GROUP BY` 子句通过列或表达式的值将一组行分组为一个小分组的汇总行记录。

`GROUP BY` 子句为每个分组返回一行。换句话说，它减少了结果集中的行数。

经常使用 `GROUP BY` 子句与[聚合函数](#)一起使用，如[SUM](#)，[AVG](#)，[MAX](#)，[MIN](#)和[COUNT](#)。SELECT子句中使用聚合函数来计算有关每个分组的信息。

`GROUP BY` 子句是[SELECT语句](#)的可选子句。下面是 `GROUP BY` 子句语法：

```
SELECT  
    c1, c2, ..., cn, aggregate_function(ci)  
FROM  
    table  
WHERE  
    where_conditions  
GROUP BY c1 , c2, ...,cn;
```

`GROUP BY` 子句必须出现在 `FROM` 和 `WHERE` 子句之后。在 `GROUP BY` 关键字之后是一个以逗号分隔的列或表达式的列表，这些是要用作为条件来对行进行分组。

MySQL GROUP BY示例

简单的MySQL GROUP BY示例

我们来看看示例数据库([yiibaidb](#))中的 `orders` 表，其结构如下所示 -

```
mysql> desc orders;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| orderNumber | int(11) | NO | PRI | NULL |
| orderDate | date | NO | | NULL |
| requiredDate | date | NO | | NULL |
| shippedDate | date | YES | | NULL |
| status | varchar(15) | NO | | NULL |
| comments | text | YES | | NULL |
| customerNumber | int(11) | NO | MUL | NULL |
+-----+-----+-----+-----+-----+
7 rows in set
```

假设要将订单状态的值分组到子组中，则要使用 `GROUP BY` 子句并指定按 `status` 列来执行分组，如下查询：

```
SELECT
    status
FROM
    orders
GROUP BY status;
```

执行上面查询语句，得到以下结果 -

```
+-----+
| status |
+-----+
| Cancelled |
| Disputed |
| In Process |
| On Hold |
| Resolved |
| Shipped |
+-----+
```

可以看到，`GROUP BY` 子句返回状态(`status`)值是唯一的。它像`DISTINCT`运算符一样工作，如下面的查询所示：

```
SELECT DISTINCT  
    status  
FROM  
    orders;
```

执行上面查询语句，得到以下结果 -

status
Shipped
Resolved
Cancelled
On Hold
Disputed
In Process

MySQL GROUP BY与聚合函数

可使用[聚合函数](#)来执行一组行的计算并返回单个值。[GROUP BY 子句](#)通常与聚合函数一起使用以执行计算每个分组并返回单个值。

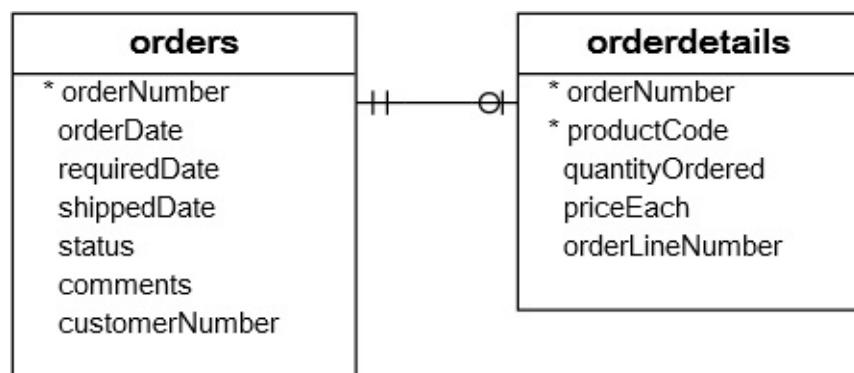
例如，如果想知道每个状态中的订单数，可以使用 [COUNT 函数](#)与 [GROUP BY 子句](#)查询语句，如下所示：

```
SELECT  
    status, COUNT(*) AS total_number  
FROM  
    orders  
GROUP BY status;
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| status | total_number |
+-----+-----+
| Cancelled | 6 |
| Disputed | 3 |
| In Process | 6 |
| On Hold | 4 |
| Resolved | 4 |
| Shipped | 303 |
+-----+
6 rows in set
```

请参阅以下订单(`orders`)和订单详细(`orderdetails`)表，它们的ER图如下所示 -



要按状态获取所有订单的总金额，可以使用 `orderdetails` 表连接 `orders` 表，并使用 `SUM` 函数计算总金额。请参阅以下查询：

```

SELECT
    status, SUM(quantityOrdered * priceEach) AS amount
FROM
    orders
        INNER JOIN
    orderdetails USING (orderNumber)
GROUP BY status;
```

执行上面查询，得到以下结果 -

status	amount
Cancelled	238854.18
Disputed	61158.78
In Process	135271.52
On Hold	169575.61
Resolved	134235.88
Shipped	8865094.64

类似地，以下查询返回订单号和每个订单的总金额。

```
SELECT
    orderNumber,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orderdetails
GROUP BY orderNumber;
```

执行上面查询，得到以下结果 -

orderNumber	total
10100	10223.83
10101	10549.01
10102	5494.78
10103	50218.95
10104	40206.20
10105	53959.21
...	...
...	这里省略了一大波数据 ...
10423	8597.73
10424	29310.30
10425	41623.44
326 rows in set	

MySQL GROUP BY用表达式示例

除了列之外，可以按表达式对行进行分组。以下查询获取每年的总销售额。

```
SELECT
    YEAR(orderDate) AS year,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orders
    INNER JOIN
    orderdetails USING (orderNumber)
WHERE
    status = 'Shipped'
GROUP BY YEAR(orderDate);
```

执行上面查询，得到以下结果

year	total
2013	3223095.80
2014	4300602.99
2015	1341395.85

在这个例子中，我们使用YEAR函数从订单日期(orderDate)中提取年份数据。只包括已发货(Shipped)状态的订单。请注意，SELECT子句中出现的表达式必须与 GROUP BY 子句中的相同。

MySQL GROUP BY与HAVING子句

可使用HAVING子句过滤 GROUP BY 子句返回的分组。以下查询使用 HAVING 子句来选择 2013 年以后的年销售额。

```

SELECT
    YEAR(orderDate) AS year,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orders
        INNER JOIN
    orderdetails USING (orderNumber)
WHERE
    status = 'Shipped'
GROUP BY year
HAVING year > 2013;

```

执行上面查询，得到以下结果 -

year	total
2014	4300602.99
2015	1341395.85

GROUP BY子句：MySQL与标准SQL

标准SQL不允许使用 `GROUP BY` 子句中的别名，但MySQL支持此选项。以下查询从订单日期提取年份，并对每年的订单进行计数。该 `year` 用作表达式 `YEAR(orderDate)` 的别名，它也用作 `GROUP BY` 子句中的别名，此查询在标准SQL中无效。参考以下查询 -

```

SELECT
    YEAR(orderDate) AS year, COUNT(orderNumber)
FROM
    orders
GROUP BY year;

```

执行上面查询，得到以下结果 -

year	COUNT(orderNumber)
2013	111
2014	151
2015	64

MySQL还允许您以升序或降序(标准SQL不能提供)对组进行排序。默认顺序是升序。例如，如果要按状态获取订单数量并按降序对状态进行排序，则可以使用带有 `DESC` 的 `GROUP BY` 子句，如下查询语句：

```
SELECT
    status, COUNT(*)
FROM
    orders
GROUP BY status DESC;
```

执行上面查询，得到以下结果 -

status	COUNT(*)
Shipped	303
Resolved	4
On Hold	4
In Process	6
Disputed	3
Cancelled	6

请注意，在 `GROUP BY` 子句中使用 `DESC` 以降序对状态进行排序。我们还可以在 `GROUP BY` 子句中明确指定 `ASC`，按状态对分组进行升序排序。

在本教程中，我们向您演示了如何使用MySQL `GROUP BY` 子句根据列或表达式的值将行分组到子组中。

在本教程中，您将学习如何使用MySQL `HAVING` 子句为行分组或聚合组指定过滤条件。

MySQL HAVING子句简介

在[SELECT语句](#)中使用 `HAVING` 子句来指定一组行或聚合的过滤条件。

`HAVING` 子句通常与[GROUP BY](#)子句一起使用，以根据指定的条件过滤分组。如果省略 `GROUP BY` 子句，则 `HAVING` 子句的行为与 `WHERE` 子句类似。

请注意，`HAVING` 子句将过滤条件应用于每组分行，而 `WHERE` 子句将过滤条件应用于每个单独的行。

MySQL HAVING子句示例

让我们举一些使用 `HAVING` 子句的例子来看看它是如何工作。我们将使用[示例数据库\(yiibaidb\)](#)中的 `orderdetails` 表进行演示。

```
mysql> desc orderdetails;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| orderNumber | int(11) | NO | PRI | NULL |
| productCode | varchar(15) | NO | PRI | NULL |
| quantityOrdered | int(11) | NO | | NULL |
| priceEach | decimal(10,2) | NO | | NULL |
| orderLineNumber | smallint(6) | NO | | NULL |
+-----+-----+-----+-----+-----+
5 rows in set
```

6.2 HAVING 子句

可以使用 `GROUP BY` 子句来获取订单号，查看每个订单销售的商品数量和每个销售总额：

```
SELECT
    ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM
    orderdetails
GROUP BY ordernumber;
```

执行上面查询语句，得到以下结果(部分) -

ordernumber	itemsCount	total
10100	151	10223.83
10101	142	10549.01
10102	80	5494.78
10103	541	50218.95
10104	443	40206.20
10105	545	53959.21
10106	675	52151.81
----- 这里省略了一大波数据 -----		
10421	75	7639.10
10422	76	5849.44
10423	111	8597.73
10424	269	29310.30
10425	427	41623.44

326 rows in set

现在，可以通过使用 `HAVING` 子句查询(过滤)哪些订单的总销售额大于 55000 ，如下所示：

```

SELECT
    ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM
    orderdetails
GROUP BY ordernumber
HAVING total > 55000;

```

执行上面查询语句，得到以下结果 -

ordernumber	itemsCount	total
10126	617	57131.92
10127	540	58841.35
10135	607	55601.84
10142	577	56052.56
10165	670	67392.85
10181	522	55069.55
10192	585	55425.77
10204	619	58793.53
10207	615	59265.14
10212	612	59830.55
10222	717	56822.65
10287	595	61402.00
10310	619	61234.67
10312	601	55639.66
10390	603	55902.50

可以使用逻辑运算符(如 OR 和 AND)在 HAVING 子句中构造复杂过滤条件。假设您想查找哪些订单的总销售额大于 50000，并且包含超过 600 个项目，则可以使用以下查询：

```

SELECT
    ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM
    orderdetails
GROUP BY ordernumber
HAVING total > 50000 AND itemsCount > 600;

```

执行上面查询语句，得到以下结果 -

ordernumber	itemsCount	total
10106	675	52151.81
10126	617	57131.92
10135	607	55601.84
10165	670	67392.85
10168	642	50743.65
10204	619	58793.53
10207	615	59265.14
10212	612	59830.55
10222	717	56822.65
10310	619	61234.67
10312	601	55639.66
10360	620	52166.00
10390	603	55902.50
10414	609	50806.85

假设您想查找所有已发货(`status='Shiped'`)的订单和总销售额大于 55000 的订单，可以使用**INNER JOIN**子句将 `orders` 表与 `orderdetails` 表一起使用，并在 `status` 列和总金额(`total`)列上应用条件，如以下查询所示：

执行上面查询，得到以下结果 -

ordernumber	status	total
10126	Shipped	57131.92
10127	Shipped	58841.35
10135	Shipped	55601.84
10142	Shipped	56052.56
10165	Shipped	67392.85
10181	Shipped	55069.55
10192	Shipped	55425.77
10204	Shipped	58793.53
10207	Shipped	59265.14
10212	Shipped	59830.55
10222	Shipped	56822.65
10287	Shipped	61402.00
10310	Shipped	61234.67
10312	Shipped	55639.66
10390	Shipped	55902.50

HAVING 子句仅在使用 GROUP BY 子句生成高级报告的输出时才有用。例如，您可以使用 HAVING 子句来回答统计问题，例如在本月，本季度或今年总销售额超过 10000 的订单。

在本教程中，您已经学习了如何使用具有 GROUP BY 子句的 MySQL HAVING 子句为行分组或聚合分组指定过滤器条件。

在本教程中，您将学习如何使用MySQL子查询编写复杂的查询语句并解释相关的子查询概念。

MySQL子查询是嵌套在另一个查询(如SELECT，INSERT，UPDATE或DELETE)中的查询。另外，MySQL子查询可以嵌套在另一个子查询中。

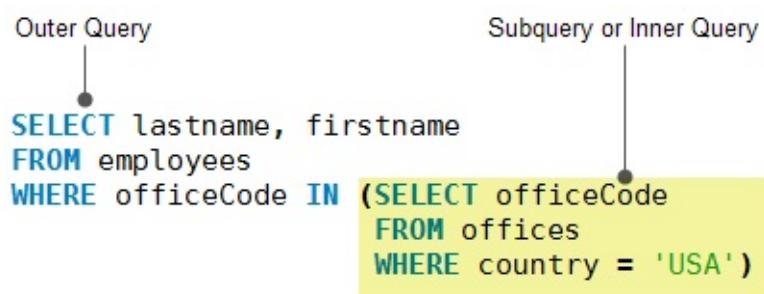
MySQL子查询称为内部查询，而包含子查询的查询称为外部查询。子查询可以在使用表达式的任何地方使用，并且必须在括号中关闭。

以下查询返回在位于美国(USA)的办公室工作的员工。

```
SELECT
    lastName, firstName
FROM
    employees
WHERE
    officeCode IN (SELECT
                    officeCode
                FROM
                    offices
                WHERE
                    country = 'USA');
```

在这个例子中：

- 子查询返回位于美国的办公室的所有办公室代码。
- 外部查询选择在办公室代码在子查询返回的结果集中的办公室中工作的员工的姓氏和名字。



当查询执行时，首先执行子查询并返回一个结果集。然后，将此结果集作为外部查询的输入。

1. MySQL子查询在WHERE子句中

我们将使用示例数据(yiibaidb)库中的 payments 表进行演示。payments 表的表结构如下 -

```
mysql> desc payments;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| customerNumber | int(11) | NO | PRI | NULL |
| checkNumber | varchar(50) | NO | PRI | NULL |
| paymentDate | date | NO | | NULL |
| amount | decimal(10, 2) | NO | | NULL |
+-----+-----+-----+-----+-----+
4 rows in set
```

1.1 MySQL子查询与比较运算符

可以使用比较运算符，例如 = , > , < 等将子查询返回的单个值与 WHERE 子句中的表达式进行比较。

例如，以下查询返回最大付款额的客户。

```

SELECT
    customerNumber, checkNumber, amount
FROM
    payments
WHERE
    amount = (SELECT
        MAX(amount)
    FROM
        payments);

```

执行上面查询语句，得到以下结果 -

customerNumber	checkNumber	amount
141	JE105477	120166.58

除等式运算符之外，还可以使用大于($>$)，小于($<$)等的其他比较运算符。

例如，可以使用子查询找到其付款大于平均付款的客户。首先，使用子查询来计算使用**AVG**聚合函数的平均付款。然后，在外部查询中，查询大于子查询返回的平均付款的付款。参考以下查询语句的写法 -

```

SELECT
    customerNumber, checkNumber, amount
FROM
    payments
WHERE
    amount > (SELECT
        AVG(amount)
    FROM
        payments);

```

执行上面查询语句，得到以下结果 -

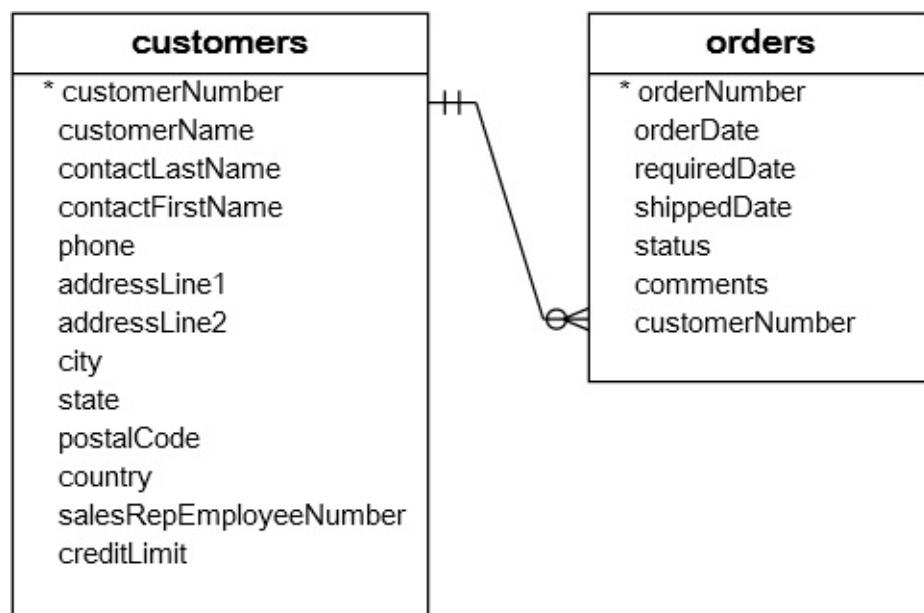
customerNumber	checkNumber	amount
112	HQ55022	32641.98
112	ND748579	33347.88
114	GG31455	45864.03
114	MA765515	82261.22
114	NR27552	44894.74
119	LN373447	47924.19
119	NG94694	49523.67
省略部分数据		
484	JH546765	47513.19
486	HS86661	45994.07
495	BH167026	59265.14
496	MN89921	52166

134 rows in set

1.2. 具有IN和NOT IN运算符的MySQL子查询

如果子查询返回多个值，则可以在 WHERE 子句中使用 IN 或 NOT IN 运算符等其他运算符。

查看以下客户和订单表的ER结构图 -



例如，可以使用带有 NOT IN 运算符的子查询来查找没有下过任何订单的客户，如下所示：

```
SELECT
    customerName
FROM
    customers
WHERE
    customerNumber NOT IN (SELECT DISTINCT
        customerNumber
    FROM
        orders);
```

执行上面查询，得到以下结果 -

```
+-----+
| customerName
+-----+
| Havel & Zbyszek Co
| American Souvenirs Inc
| Porto Imports Co.
| Asian Shopping Network, Co
| Natrlich Autos
| ANG Resellers
| Messner Shopping Network
| Franken Gifts, Co
| BG&E Collectables
| Schuyler Imports
| Der Hund Imports
| Cramer Spezialitten, Ltd
| Asian Treasures, Inc.
| SAR Distributors, Co
| Kommission Auto
| Lisboa Souveniers, Inc
| Precious Collectables
| Stuttgart Collectable Exchange
| Feuer Online Stores, Inc
| Warburg Exchange
| Anton Designs, Ltd.
| Mit Vergngen & Co.
| Kremlin Collectables, Co.
| Raanan Stores, Inc
+-----+
24 rows in set
```

3. FROM子句中的MySQL子查询

在FROM子句中使用子查询时，从子查询返回的结果集将用作[临时表](#)。该表称为[派生表](#)或物化子查询。

以下子查询将查找订单表中的[最大](#)，[最小](#)和[平均数](#)：

```

SELECT
    MAX(items), MIN(items), FLOOR(AVG(items))
FROM
    (SELECT
        orderNumber, COUNT(orderNumber) AS items
    FROM
        orderdetails
    GROUP BY orderNumber) AS lineitems;

```

执行上面查询，得到以下结果 -

MAX(items)	MIN(items)	FLOOR(AVG(items))
18	1	9

1 row in set

4. MySQL相关子查询

在前面的例子中，注意到一个子查询是独立的。这意味着您可以将子查询作为独立查询执行，例如：

```

SELECT
    orderNumber,
    COUNT(orderNumber) AS items
FROM
    orderdetails
GROUP BY orderNumber;

```

与独立子查询不同，相关子查询是使用外部查询中的数据的子查询。换句话说，相关的子查询取决于外部查询。对外部查询中的每一行对相关子查询进行一次评估。

在以下查询中，我们查询选择购买价格高于每个产品线中的产品的平均购买价格的产品。

```

SELECT
    productname,
    buyprice
FROM
    products p1
WHERE
    buyprice > (SELECT
                    AVG(buyprice)
                FROM
                    products
                WHERE
                    productline = p1.productline);

```

执行上面查询，得到以下结果 -

productname	buyprice
1952 Alpine Renault 1300	98.58
1996 Moto Guzzi 1100i	68.99
2003 Harley Davidson Eagle Drag Bike	91.02
1972 Alfa Romeo GTA	85.68
1962 Lancia Delta 16V	103.42
1968 Ford Mustang	95.34
2001 Ferrari Enzo	95.59
***** 此处省略了一大波数据 *****	
American Airlines: B767 300	51.15
America West Airlines B757-200	68.8
ATA: B757-300	59.33
F/A 18 Hornet 1/72	54.4
The Titanic	51.09
The Queen Mary	53.63
+-----+	
55 rows in set	

对于变化的每一行产品线，每个产品线都会执行内部查询。因此，平均购买价格也会改变。外部查询仅筛选购买价格大于子查询中每个产品线的平均购买价格的产品。

5. MySQL子查询与 EXISTS 和 NOT EXISTS

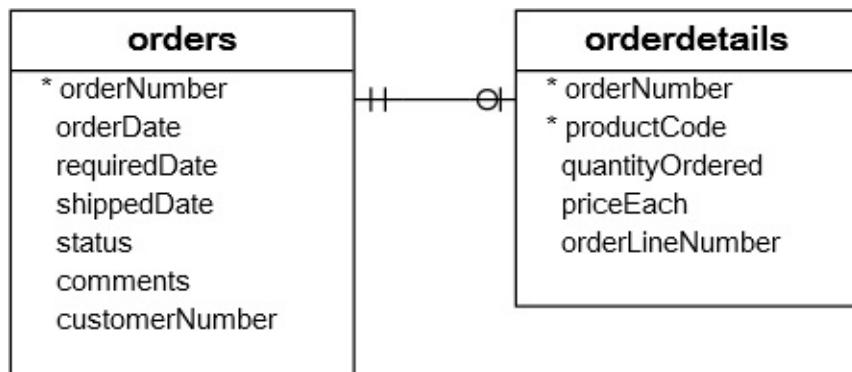
当子查询与 **EXISTS** 或 **NOT EXISTS** 运算符一起使用时，子查询返回一个布尔值为 **TRUE** 或 **FALSE** 的值。以下查询说明了与 **EXISTS** 运算符一起使用的子查询：

```
SELECT
*
FROM
table_name
WHERE
EXISTS( subquery );
```

在上面的查询中，如果子查询(**subquery**)有返回任何行，则 **EXISTS** 子查询返回 **TRUE**，否则返回 **FALSE**。

通常在相关子查询中使用 **EXISTS** 和 **NOT EXISTS**。

下面我们来看看示例数据库(yiibaidb)中的 **orders** 和 **orderDetails** 表：



以下查询选择总额大于 60000 的销售订单。

```
SELECT
    orderNumber,
    SUM(priceEach * quantityOrdered) total
FROM
    orderdetails
    INNER JOIN
    orders USING (orderNumber)
GROUP BY orderNumber
HAVING SUM(priceEach * quantityOrdered) > 60000;
```

执行上面查询，得到以下结果 -

orderNumber	total
10165	67392.85
10287	61402.00
10310	61234.67

如上面所示，返回 3 行数据，这意味着有 3 个销售订单的总额大于 60000 。

可以使用上面的查询作为相关子查询，通过使用 EXISTS 运算符来查找至少有一个总额大于 60000 的销售订单的客户信息：

```

SELECT
    customerNumber,
    customerName
FROM
    customers
WHERE
    EXISTS( SELECT
        orderNumber, SUM(priceEach * quantityOrdered)
    FROM
        orderdetails
        INNER JOIN
        orders USING (orderNumber)
    WHERE
        customerNumber = customers.customerNumber
    GROUP BY orderNumber
    HAVING SUM(priceEach * quantityOrdered) > 60000);

```

执行上面查询，得到以下结果 -

```
+-----+-----+
| customerNumber | customerName |
+-----+-----+
| 148 | Dragon Souveniers, Ltd. |
| 259 | Toms Spezialitten, Ltd |
| 298 | Vida Sport, Ltd |
+-----+
3 rows in set
```

在本教程中，我们向您演示了如何使用MySQL子查询和相关子查询来构建更复杂的查询。

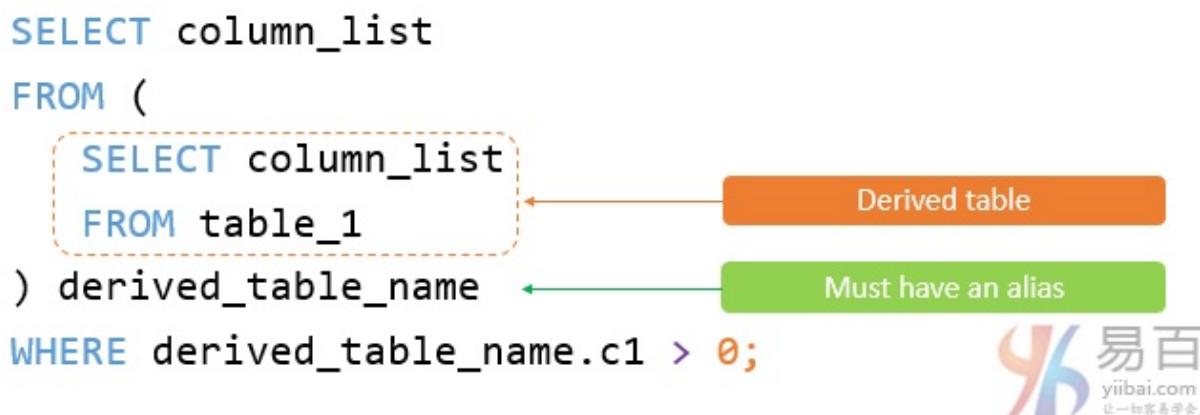
在本教程中，您将了解和学习MySQL派生表以及如何简化复杂查询。

1. MySQL派生表介绍

派生表是从SELECT语句返回的虚拟表。派生表类似于临时表，但是在SELECT语句中使用派生表比临时表简单得多，因为它不需要创建临时表的步骤。

术语：“派生表”和“子查询”通常可互换使用。当SELECT语句的FROM子句中使用独立子查询时，我们将其称为派生表。

以下说明了使用派生表的查询：



请注意，独立子查询是一个子查询，可独立于包含该语句的执行语句。

与子查询不同，派生表必须具有别名，以便稍后在查询中引用其名称。如果派生表没有别名，MySQL将发出以下错误：

Every derived table must have its own alias.

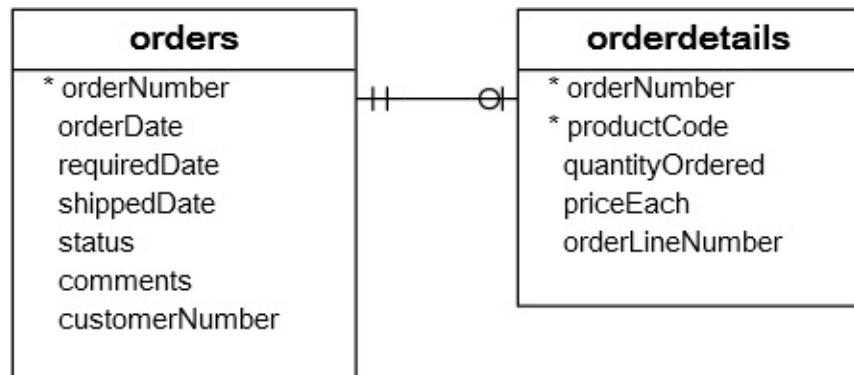
以下说明了使用派生表的SQL语句：

```

SELECT
    column_list
FROM
    (SELECT
        column_list
    FROM
        table_1) derived_table_name;
WHERE derived_table_name.c1 > 0;
  
```

2. 简单的MySQL派生表示例

以下查询从示例数据库(yiibaidb)中的 orders 表和 orderdetails 表中获得 2013 年销售收入最高的前 5 名产品：



参考以下查询语句 -

```

SELECT
    productCode,
    ROUND(SUM(quantityOrdered * priceEach)) sales
FROM
    orderdetails
    INNER JOIN
    orders USING (orderNumber)
WHERE
    YEAR(shippedDate) = 2013
GROUP BY productCode
ORDER BY sales DESC
LIMIT 5;
  
```

执行上面查询语句，得到以下结果 -

productCode	sales
S18_3232	103480
S10_1949	67985
S12_1108	59852
S12_3891	57403
S12_1099	56462

5 rows in set

您可以使用此查询的结果作为派生表，并将其与 `products` 表相关联，`products` 表的结构如下所示：

```
mysql> desc products;
+-----+-----+-----+-----+-----+
| Field          | Type       | Null | Key  | Default | Ex |
+-----+-----+-----+-----+-----+
| tra           | varchar(15) | NO   | PRI  |          |    |
+-----+-----+-----+-----+-----+
| productCode   | varchar(15) | NO   | PRI  |          |    |
| productName   | varchar(70)  | NO   |       | NULL    |    |
| productLine   | varchar(50)  | NO   | MUL  |          |    |
| productScale  | varchar(10)  | NO   |       | NULL    |    |
| productVendor | varchar(50)  | NO   |       | NULL    |    |
| productDescription | text      | NO   |       | NULL    |    |
| quantityInStock | smallint(6) | NO   |       | NULL    |    |
| buyPrice      | decimal(10,2) | NO   |       | NULL    |    |
| MSRP          | decimal(10,2) | NO   |       | NULL    |    |
+-----+-----+-----+-----+-----+
9 rows in set
```

参考以下查询语句 -

```

SELECT
    productName, sales
FROM
    (
        SELECT
            productCode,
            ROUND(SUM(quantityOrdered * priceEach)) sales
        FROM
            orderdetails
        INNER JOIN orders USING (orderNumber)
        WHERE
            YEAR(shippedDate) = 2013
        GROUP BY productCode
        ORDER BY sales DESC
        LIMIT 5) top5products2013
INNER JOIN
    products USING (productCode);

```

执行上面查询语句，得到以下结果 -

productName	sales
1992 Ferrari 360 Spider red	103480
1952 Alpine Renault 1300	67985
2001 Ferrari Enzo	59852
1969 Ford Falcon	57403
1968 Ford Mustang	56462

5 rows in set

在上面这个例子中：

- 首先，执行子查询来创建一个结果集或派生表。
- 然后，在 `productCode` 列上使用 `products` 表连接 `top5product2013` 派生表的外部查询。

3. 一个更复杂的MySQL派生表示例

假设必须将 2013 年的客户分为 3 组：铂金，白金和白银。此外，需要了解每个组中的客户数量，具体情况如下：

- 订单总额大于 100000 的为铂金客户；
- 订单总额为 10000 至 100000 的为黄金客户
- 订单总额为小于 10000 的为银牌客户

要构建此查询，首先，您需要使用 CASE 表达式和 GROUP BY 子句将每个客户放入相应的分组中，如下所示：

```
SELECT
    customerNumber,
    ROUND(SUM(quantityOrdered * priceEach)) sales,
    (CASE
        WHEN SUM(quantityOrdered * priceEach) < 10000 THEN 'Silver'
        WHEN SUM(quantityOrdered * priceEach) BETWEEN 10000 AND 100000 THEN 'Gold'
        WHEN SUM(quantityOrdered * priceEach) > 100000 THEN 'Platinum'
    END) customerGroup
FROM
    orderdetails
    INNER JOIN
    orders USING (orderNumber)
WHERE
    YEAR(shippedDate) = 2013
GROUP BY customerNumber
ORDER BY sales DESC;
```

以下是查询的输出：

customerNumber	sales	customerGroup
141	189840	Platinum
124	167783	Platinum
148	150123	Platinum
151	117635	Platinum
320	93565	Gold
278	89876	Gold
161	89419	Gold
*****此处省略了一大波数据*****		
219	4466	Silver
323	2880	Silver
381	2756	Silver

然后，可以使用此查询作为派生表，并按如下所示进行分组：

```

SELECT
    customerGroup,
    COUNT(CG.customerGroup) AS groupCount
FROM
    (SELECT
        customerNumber,
        ROUND(SUM(quantityOrdered * priceEach)) sales,
        (CASE
            WHEN SUM(quantityOrdered * priceEach) < 10000 THEN 'Silver'
            WHEN SUM(quantityOrdered * priceEach) BETWEEN 10000 AND 100000 THEN 'Gold'
            WHEN SUM(quantityOrdered * priceEach) > 100000 THEN 'Platinum'
        END) customerGroup
    FROM
        orderdetails
    INNER JOIN orders USING (orderNumber)
    WHERE
        YEAR(shippedDate) = 2013
    GROUP BY customerNumber) CG
GROUP BY CG.customerGroup;

```

执行上面查询语句，得到以下结果 -

customerGroup	groupCount
Gold	61
Platinum	4
Silver	8
3 rows in set	

在本教程中，您已经学会了如何使用 `FROM` 子句中的子查询作为MySQL派生表来简化复杂查询。

在本教程中，您将学习如何使用MySQL CTE或公用表表达式以更可读的方式构建复杂查询。

自MySQL 8.0版以来简要介绍了公共表表达式或叫CTE的功能，因此需要您在计算机上安装MySQL 8.0，以便在本教程中练习本语句。

1. 什么是公用表表达式或CTE？

公用表表达式是一个命名的临时结果集，仅在单个SQL语句(例如 `SELECT`，`INSERT`，`UPDATE`或`DELETE`)的执行范围内存在。

与派生表类似，CTE不作为对象存储，仅在查询执行期间持续。与派生表不同，CTE可以是自引用(递归CTE)，也可以在同一查询中多次引用。此外，与派生表相比，CTE提供了更好的可读性和性能。

2. MySQL CTE语法

CTE的结构包括名称，可选列列表和定义CTE的查询。定义CTE后，可以像 `SELECT`，`INSERT`，`UPDATE`，`DELETE` 或 `CREATE VIEW` 语句中的视图一样使用它。

以下说明了CTE的基本语法：

```
WITH cte_name (column_list) AS (
    query
)
SELECT * FROM cte_name;
```

请注意，查询中的列数必须与 `column_list` 中的列数相同。如果省略 `column_list`，CTE将使用定义CTE的查询的列列表。

3. 简单的MySQL CTE示例

以下示例说明如何使用CTE查询[示例数据库\(yiibaidb\)](#)中的 `customers` 表中的数据。请注意，此示例仅用于演示目的，以便您更容易地了解CTE概念。

```

WITH customers_in_usa AS (
    SELECT
        customerName, state
    FROM
        customers
    WHERE
        country = 'USA'
) SELECT
    customerName
FROM
    customers_in_usa
WHERE
    state = 'CA'
ORDER BY customerName;

```

注意：上面语句只能在 MySQL8.0 以上版本才支持。

执行上面查询语句，得到以下结果(部分)

	customerName
▶	Boards & Toys Co.
	Collectable Mini Designs Co.
	Corporate Gift Ideas Co.
	Men 'R' US Retailers, Ltd.
	Mini Gifts Distributors Ltd.
	Mini Wheels Co.
	Signal Collectibles Ltd.
	Technics Stores Inc.
	The Sharp Gifts Warehouse
	Toys4GrownUps.com
	West Coast Collectables Co.

在此示例中，CTE的名称为 `customers_in_usa`，定义CTE的查询返回两列：`customerName` 和 `state`。因此，`customers_in_usa` CTE返回位于美国的所有客户。

在定义美国CTE的客户之后，我们可在 `SELECT` 语句中引用它，例如，仅查询选择位于*California* 的客户。

参见另外一个例子：

```

WITH topsales2013 AS (
    SELECT
        salesRepEmployeeNumber employeeNumber,
        SUM(quantityOrdered * priceEach) sales
    FROM
        orders
        INNER JOIN
        orderdetails USING (orderNumber)
        INNER JOIN
        customers USING (customerNumber)
    WHERE
        YEAR(shippedDate) = 2013
        AND status = 'Shipped'
    GROUP BY salesRepEmployeeNumber
    ORDER BY sales DESC
    LIMIT 5
)
SELECT
    employeeNumber, firstName, lastName, sales
FROM
    employees
    JOIN
    topsales2013 USING (employeeNumber);

```

执行上面查询后，得到以下结果 -

	employeeNumber	firstName	lastName	sales
▶	1165	Leslie	Jennings	413219.85
	1370	Gerard	Hernandez	295246.44
	1401	Pamela	Castillo	289982.88
	1621	Mami	Nishi	267249.40
	1501	Larry	Bott	261536.95

在这个例子中，CTE中返回了在2013年前五名的销售代表。之后，我们引用了 topsales2013 CTE来获取有关销售代表的其他信息，包括名字和姓氏。

4. 更高级的MySQL CTE示例

请参阅以下示例：

```

WITH salesrep AS (
    SELECT
        employeeNumber,
        CONCAT(firstName, ' ', lastName) AS salesrepName
    FROM
        employees
    WHERE
        jobTitle = 'Sales Rep'
),
customer_salesrep AS (
    SELECT
        customerName, salesrepName
    FROM
        customers
        INNER JOIN
        salesrep ON employeeNumber = salesrepEmployeeNumber
)
SELECT
    *
FROM
    customer_salesrep
ORDER BY customerName;

```

执行上面查询语句，得到以下结果 -

	customerName	salesrepName
▶	Alpha Cognac	Gerard Hernandez
	American Souvenirs Inc	Foon Yue Tseng
	Amica Models & Co.	Pamela Castillo
	Anna's Decorations, Ltd	Andy Fixter
	Atelier graphique	Gerard Hernandez
	Australian Collectables, Ltd	Andy Fixter
	Australian Collectors, Co.	Andy Fixter
	Australian Gift Network, Co	Andy Fixter
	Auto Associés & Cie.	Gerard Hernandez
	Auto Canal+ Petit	Loui Bondur
	Auto-Moto Classics Inc.	Steve Patterson
	AV Stores, Co.	Larry Bott

在这个例子中，在同一查询中有两个CTE。第一个CTE(salesrep)获得职位是销售代表的员工。第二个CTE(customer_salesrep)使用 INNER JOIN 子句与第一个CTE连接来获取每个销售代表负责的客户。

在使用第二个`CTE`之后，使用带有`ORDER BY`子句的简单`SELECT`语句来查询来自该`CTE`的数据。

5. WITH子句用法

有一些上下文可以使用`WITH`子句来创建公用表表达式(`CTE`)：

首先，在`SELECT`，`UPDATE`和`DELETE`语句的开头可以使用`WITH`子句：

```
WITH ... SELECT ...
WITH ... UPDATE ...
WITH ... DELETE ...
```

第二，可以在子查询或派生表子查询的开头使用`WITH`子句：

```
SELECT ... WHERE id IN (WITH ... SELECT ...);
SELECT * FROM (WITH ... SELECT ...) AS derived_table;
```

第三，可以在`SELECT`语句之前立即使用`WITH`子句，包括`SELECT`子句：

```
CREATE TABLE ... WITH ... SELECT ...
CREATE VIEW ... WITH ... SELECT ...
INSERT ... WITH ... SELECT ...
REPLACE ... WITH ... SELECT ...
DECLARE CURSOR ... WITH ... SELECT ...
EXPLAIN ... WITH ... SELECT ...
```

在本教程中，您已经学会了如何使用MySQL公共表表达式(`CTE`)来构造复杂的查询语句。

在本教程中，您将了解MySQL递归CTE(公共表表达式)以及如何使用它来遍历分层数据。

自MySQL 8.0版以来简要介绍了公共表表达式或叫CTE的功能，因此需要您在计算机上安装MySQL 8.0，以便在本教程中练习本语句。

1. MySQL递归CTE简介

递归公用表表达式(CTE)是一个具有引用CTE名称本身的子查询的CTE。以下说明递归CTE的语法 -

```
WITH RECURSIVE cte_name AS (
    initial_query -- anchor member
    UNION ALL
    recursive_query -- recursive member that references to the C
    TE name
)
SELECT * FROM cte_name;
```

递归CTE由三个主要部分组成：

- 形成CTE结构的基本结果集的初始查询(*initial_query*)，初始查询部分被称为锚成员。
- 递归查询部分是引用CTE名称的查询，因此称为递归成员。递归成员由一个UNION ALL或 UNION DISTINCT运算符与锚成员相连。
- 终止条件是当递归成员没有返回任何行时，确保递归停止。

递归CTE的执行顺序如下：

1. 首先，将成员分为两个：锚点和递归成员。
2. 接下来，执行锚成员形成基本结果集(R_0)，并使用该基本结果集进行下一次迭代。
3. 然后，将 R_i 结果集作为输入执行递归成员，并将 R_{i+1} 作为输出。
4. 之后，重复第三步，直到递归成员返回一个空结果集，换句话说，满足终止条件。
5. 最后，使用 UNION ALL 运算符将结果集从 R_0 到 R_n 组合。

2. 递归成员限制

递归成员不能包含以下结构：

- 聚合函数，如 `MAX`, `MIN`, `SUM`, `AVG`, `COUNT` 等
- `GROUP BY` 子句
- `ORDER BY` 子句
- `LIMIT` 子句
- `DISTINCT`

请注意，上述约束不适用于锚定成员。另外，只有在使用 `UNION` 运算符时，要禁止 `DISTINCT` 才适用。如果使用 `UNION DISTINCT` 运算符，则允许使用 `DISTINCT`。

另外，递归成员只能在其子句中引用 `CTE` 名称，而不是引用任何 `子查询`。

3. 简单的 MySQL 递归 CTE 示例

请参阅以下简单的递归 `CTE` 示例：

```
WITH RECURSIVE cte_count (n)
AS (
    SELECT 1
    UNION ALL
    SELECT n + 1
    FROM cte_count
    WHERE n < 3
)
SELECT n
FROM cte_count;
```

在此示例中，以下查询：

```
SELECT 1
```

是作为基本结果集返回 `1` 的锚成员。

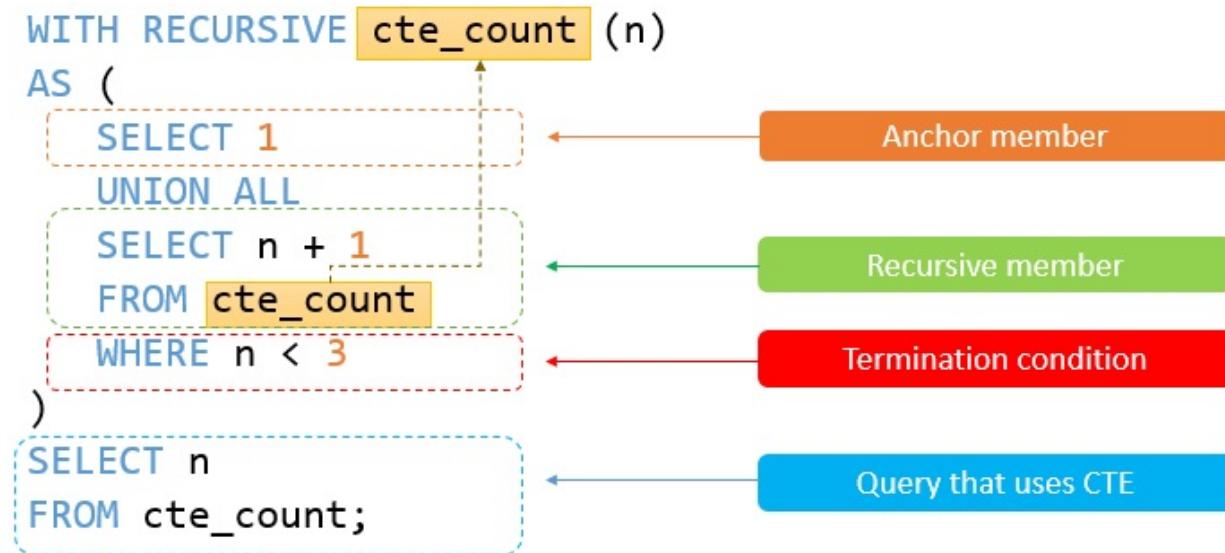
以下查询 -

```
SELECT n + 1
FROM cte_count
WHERE n < 3
```

是递归成员，因为它引用了 `cte_count` 的 **CTE** 名称。

递归成员中的表达式 `<3` 是终止条件。当 `n` 等于 `3`，递归成员将返回一个空集合，将停止递归。

下图显示了上述 **CTE** 的元素：



递归 **CTE** 返回以下输出：

	n
▶	1
	2
	3

递归 **CTE** 的执行步骤如下：

- 首先，分离锚和递归成员。
- 接下来，锚定成员形成初始行(`SELECT 1`)，因此第一次迭代在 `n = 1` 时产生 `1 + 1 = 2`。
- 然后，第二次迭代对第一次迭代的输出(`2`)进行操作，并且在 `n = 2` 时产生 `2 + 1 = 3`。
- 之后，在第三次操作(`n = 3`)之前，满足终止条件(`n < 3`)，因此查询停止。
- 最后，使用 `UNION ALL` 运算符组合所有结果集 `1, 2` 和 `3`。

4. 使用MySQL递归CTE遍历分层数据

我们将使用示例数据库(yiibaidb)中的 employees 表进行演示。

```
mysql> desc employees;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| employeeNumber | int(11) | NO | PRI | NULL |
| lastName | varchar(50) | NO | | NULL |
| firstName | varchar(50) | NO | | NULL |
| extension | varchar(10) | NO | | NULL |
| email | varchar(100) | NO | | NULL |
| officeCode | varchar(10) | NO | MUL | NULL |
| reportsTo | int(11) | YES | MUL | NULL |
| jobTitle | varchar(50) | NO | | NULL |
+-----+-----+-----+-----+-----+
8 rows in set
```

employees 表具有引用 employeeNumber 字段的 reportsTo 字段。

reportsTo 列存储经理的 ID 。总经理不会向公司的组织结构中的任何人报告，因此 reportsTo 列中的值为NULL。

您可以应用递归CTE以自顶向下的方式查询整个组织结构，如下所示：

```

WITH RECURSIVE employee_paths AS
( SELECT employeeNumber,
         reportsTo managerNumber,
         officeCode,
         1 lvl
   FROM employees
  WHERE reportsTo IS NULL
    UNION ALL
    SELECT e.employeeNumber,
           e.reportsTo,
           e.officeCode,
           lvl+1
      FROM employees e
     INNER JOIN employee_paths ep ON ep.employeeNumber = e.reportsTo )
SELECT employeeNumber,
       managerNumber,
       lvl,
       city
  FROM employee_paths ep
 INNER JOIN offices o USING (officeCode)
 ORDER BY lvl, city;

```

让我们将查询分解成更小的部分，使其更容易理解。首先，使用以下查询形成锚成员：

```

SELECT
  employeeNumber, reportsTo managerNumber, officeCode
FROM
  employees
WHERE
  reportsTo IS NULL

```

此查询(锚成员)返回 `reportTo` 为 `NULL` 的总经理。

其次，通过引用`CTE`名称来执行递归成员，在这个示例中为 `employee_paths`：

```

SELECT
    e.employeeNumber, e.reportsTo, e.officeCode
FROM
    employees e
        INNER JOIN
    employee_paths ep ON ep.employeeNumber = e.reportsTo

```

此查询(递归成员)返回经理的所有直接上级，直到没有更多的直接上级。如果递归成员不返回直接上级，则递归停止。

第三，使用 `employee_paths` 的查询将 `CTE` 返回的结果集与 `offices` 表结合起来，以得到最终结果集合。

以下是查询的输出：

	employeeNumber	managerNumber	lvl	city
▶	1002	NULL	1	San Francisco
	1076	1002	2	San Francisco
	1056	1002	2	San Francisco
	1102	1056	3	Paris
	1143	1056	3	San Francisco
	1088	1056	3	Sydney
	1621	1056	3	Tokyo
	1188	1143	4	Boston
	1216	1143	4	Boston
	1504	1102	4	London
	1501	1102	4	London
	1286	1143	4	NYC
	1323	1143	4	NYC
	1401	1102	4	Paris
	1702	1102	4	Paris
	1337	1102	4	Paris
	1370	1102	4	Paris
	1166	1143	4	San Francisco

在本教程中，您已经了解了MySQL递归 `CTE` 以及如何使用它来遍历分层数据。

使用UNION，如果想从几个表选择行一前一后的所有作为一个单一的结果集，或几个集合行在单一的表中。

UNION是从MySQL4.0开始使用。本节说明如何使用它。

假设有两个表，列出潜在和实际的客户，第三个表，列出供应商购买耗材，并且希望通过从所有三个表合并名称和地址，以创建一个单一的邮件列表。UNION提供了一种方法来做到这一点。假设这三个表具有以下内容：

```
mysql> SELECT * FROM prospect;
+-----+-----+-----+
| fname | lname | addr
+-----+-----+-----+
| Peter | Jones | 482 Rush St., Apt. 402
| Bernice | Smith | 916 Maple Dr.
+-----+-----+-----+
mysql> SELECT * FROM customer;
+-----+-----+-----+
| last_name | first_name | address
+-----+-----+-----+
| Peterson | Grace | 16055 Seminole Ave.
| Smith | Bernice | 916 Maple Dr.
| Brown | Walter | 8602 1st St.
+-----+-----+-----+
mysql> SELECT * FROM vendor;
+-----+-----+
| company | street
+-----+-----+
| ReddyParts, Inc. | 38 Industrial Blvd.
| Parts to go, Ltd. | 213B Commerce Park.
+-----+-----+
```

这不要紧，如果所有的三个表都是不同的列名称。以下查询说明如何从三个表一次全部选择名称和地址：

```
mysql> SELECT fname, lname, addr FROM prospect
-> UNION
-> SELECT first_name, last_name, address FROM customer
-> UNION
-> SELECT company, '', street FROM vendor;
+-----+-----+-----+
| fname | lname | addr
+-----+-----+-----+
| Peter | Jones | 482 Rush St., Apt. 402
| Bernice | Smith | 916 Maple Dr.
| Grace | Peterson | 16055 Seminole Ave.
| Walter | Brown | 8602 1st St.
| ReddyParts, Inc. | | 38 Industrial Blvd.
| Parts to go, Ltd. | | 213B Commerce Park.
+-----+-----+-----+
```

如果想选择所有记录，包括重复的记录，请使用UNION关键字后面接一个 ALL 关键字：

```
mysql> SELECT fname, lname, addr FROM prospect
-> UNION ALL
-> SELECT first_name, last_name, address FROM customer
-> UNION
-> SELECT company, '', street FROM vendor;
+-----+-----+-----+
| fname | lname | addr
+-----+-----+-----+
| Peter | Jones | 482 Rush St., Apt. 402
| Bernice | Smith | 916 Maple Dr.
| Grace | Peterson | 16055 Seminole Ave.
| Bernice | Smith | 916 Maple Dr.
| Walter | Brown | 8602 1st St.
| ReddyParts, Inc. | | 38 Industrial Blvd.
| Parts to go, Ltd. | | 213B Commerce Park.
+-----+-----+-----+
```

在本教程中，我们将向您介绍SQL `INTERSECT` 运算符，并展示如何模拟MySQL `INTERSECT` 运算符(交集)。

1. SQL INTERSECT操作符简介

`INTERSECT` 运算符是一个集合运算符，它只返回两个查询或更多查询的交集。

以下说明 `INTERSECT` 运算符的语法。

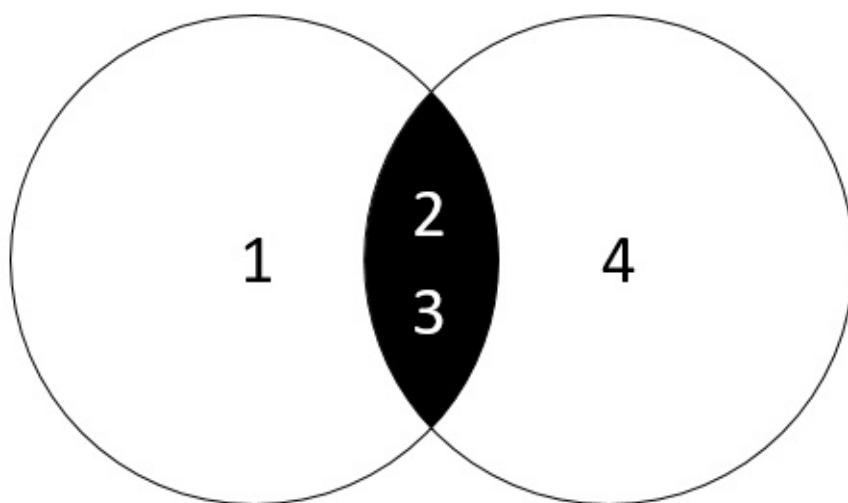
```
(SELECT column_list
FROM table_1)
INTERSECT
(SELECT column_list
FROM table_2);
```

`INTERSECT` 运算符比较两个查询的结果，并返回由左和右查询输出的不同行记录。

要将 `INTERSECT` 运算符用于两个查询，应用以下规则：

- 列的顺序和数量必须相同。
- 相应列的数据类型必须兼容或可转换。

下图说明了 `INTERSECT` 运算符。



左侧查询产生一个结果集(1 , 2 , 3)，右侧查询返回一个结果集(2 , 3 , 4)。

`INTERSECT` 操作符返回包含(2 , 3)，也就是两个结果集的相交的行记录。与 `UNION`运算符不同， `INTERSECT` 运算符返回两个集合之间的交点。

请注意，SQL标准有三个集合运算符，包括UNION，INTERSECT 和MINUS。

2. MySQL INTERSECT模拟

不幸的是，MySQL不支持 INTERSECT 操作符。但是我们可以模拟 INTERSECT 操作符。

我们为演示创建一些示例数据。

以下语句创建表 t1 和 t2 ，然后将数据插入到两个表中。

```
USE testdb;

DROP TABLE IF exists t1;
DROP TABLE IF exists t2;

CREATE TABLE t1 (
    id INT PRIMARY KEY
);

CREATE TABLE t2 LIKE t1;

INSERT INTO t1(id) VALUES(1),(2),(3);

INSERT INTO t2(id) VALUES(2),(3),(4);
```

以下从 t1 表查询返回行记录如下 -

```
mysql> SELECT id FROM t1;
+----+
| id |
+----+
| 1 |
| 2 |
| 3 |
+----+
3 rows in set
```

以下从 t2 表查询返回行记录如下 -

```
mysql> SELECT id FROM t2;
+----+
| id |
+----+
| 2 |
| 3 |
| 4 |
+----+
3 rows in set
```

使用**DISTINCT**运算符和**INNER JOIN**子句模拟**MySQL INTERSECT**运算符

以下语句使用**DISTINCT**运算符和**INNER JOIN**子句来返回两个表中的相交集合：

```
SELECT DISTINCT
    id
FROM t1
    INNER JOIN t2 USING(id);
```

执行上面查询语句，得到以下结果 -

```
+----+
| id |
+----+
| 2 |
| 3 |
+----+
2 rows in set
```

上面语句是怎么工作的？

- **INNER JOIN** 子句从左表和右表返回所有符合条件的行记录。
- **DISTINCT** 运算符删除重复行。

使用**IN**运算符和子查询模拟**MySQL INTERSECT**运算符

以下语句使用**IN**运算符和子查询返回两个结果集的交集。

```
SELECT DISTINCT
    id
FROM
    t1
WHERE
    id IN (SELECT
        id
    FROM
        t2);
```

执行以上查询语句，得到以下结果 -

```
+----+
| id |
+----+
| 2 |
| 3 |
+----+
2 rows in set
```

上面查询语句是如何工作的？

- 子查询返回第一个结果集。
- 外部查询使用 `IN` 运算符仅选择第一个结果集中的值。`DISTINCT` 运算符确保只选择不同的值。

在本教程中，您已经学习了几种方法来模拟MySQL中的 `INTERSECT` (交集)运算符。

在本教程中，您将学习如何使用MySQL `INSERT` 语句将数据插入到数据库表中。

1. 简单的MySQL `INSERT` 语句

MySQL `INSERT` 语句允许您将一行或多行插入到表中。下面说明了 `INSERT` 语句的语法：

```
INSERT INTO table(column1, column2...)
VALUES (value1, value2, ...);
```

首先，在 `INSERT INTO` 子句之后，在括号内指定表名和逗号分隔列的列表。然后，将括号内的相应列的逗号分隔值放在 `VALUES` 关键字之后。

在执行插入语句前，需要具有执行 `INSERT` 语句的 `INSERT` 权限。让我们创建一个名为 `tasks` 的新表来练习 `INSERT` 语句，参考以下创建语句 -

```
USE testdb;

CREATE TABLE IF NOT EXISTS tasks (
    task_id INT(11) AUTO_INCREMENT,
    subject VARCHAR(45) DEFAULT NULL,
    start_date DATE DEFAULT NULL,
    end_date DATE DEFAULT NULL,
    description VARCHAR(200) DEFAULT NULL,
    PRIMARY KEY (task_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

例如，如果要将任务插入到 `tasks` 表中，则使用 `INSERT` 语句如下：

```
INSERT INTO tasks(subject, start_date, end_date, description)
VALUES('Learn MySQL INSERT', '2017-07-21', '2017-07-22', 'Start learning...');
```

执行该语句后，MySQL 返回一条消息以通知受影响的行数。在这种情况下，有一行受到影响。现在使用以下语句查询 `tasks` 中的数据，如下所示 -

```
SELECT * FROM tasks;
```

执行上面查询语句，得到以下结果 -

task_id	subject	start_date	end_date	description
1	Learn MySQL INSERT	2017-07-21	2017-07-22	Start learning..

1 row in set

2. MySQL INSERT - 插入多行

想要在表中一次插入多行，可以使用具有以下语法的 `INSERT` 语句：

```
INSERT INTO table(column1, column2...)
VALUES (value1, value2, ...),
       (value1, value2, ...),
       ...;
```

在这种形式中，每行的值列表用逗号分隔。例如，要将多行插入到 `tasks` 表中，请使用以下语句：

```
INSERT INTO tasks(subject, start_date, end_date, description)
VALUES ('任务-1', '2017-01-01', '2017-01-02', 'Description 1'),
       ('任务-2', '2017-01-01', '2017-01-02', 'Description 2'),
       ('任务-3', '2017-01-01', '2017-01-02', 'Description 3');
```

执行上面语句后，返回 -

```
Query OK, 3 rows affected
Records: 3  Duplicates: 0  Warnings: 0
```

现在查询 tasks 表中的数据，如下所示 -

```
select * from tasks;
```

执行上面查询语句，得到以下结果 -

task_id	subject	start_date	end_date	description
1	Learn MySQL INSERT	2017-07-21	2017-07-22	Start learning..
2	任务 1	2017-01-01	2017-01-02	Description 1
3	任务 2	2017-01-01	2017-01-02	Description 2
4	任务 3	2017-01-01	2017-01-02	Description 3

4 rows in set

如果为表中的所有列指定相应列的值，则可以忽略 INSERT 语句中的列列表，如下所示：

```
INSERT INTO table
VALUES (value1,value2,...);
```

或者 -

```
INSERT INTO table
VALUES (value1,value2,...),
       (value1,value2,...),
       ...;
```

请注意，不必为自动递增列(例如 `taskid` 列)指定值，因为MySQL会自动为自动递增列生成值。

3. 具有SELECT子句的MySQL INSERT

在MySQL中，可以使用 `SELECT` 语句 返回的列和值来填充 `INSERT` 语句的值。此功能非常方便，因为您可以使用 `INSERT` 和 `SELECT` 子句完全或部分复制表，如下所示：

```
INSERT INTO table_1
SELECT c1, c2, FROM table_2;
```

假设要将 `tasks` 表复制到 `tasks_bak` 表。

首先，通过复制 `tasks` 表的结构，创建一个名为 `tasks_bak` 的新表，如下所示：

```
CREATE TABLE tasks_bak LIKE tasks;
```

第二步，使用以下 `INSERT` 语句将 `tasks` 表中的数据插入 `tasks_bak` 表：

```
INSERT INTO tasks_bak
SELECT * FROM tasks;
```

第三步，检查 `tasks_bak` 表中的数据，看看是否真正从 `tasks` 表复制完成了。

```
mysql> select * from tasks;
+-----+-----+-----+-----+
| task_id | subject | start_date | end_date | descr
|ption |
+-----+-----+-----+-----+
| 1 | Learn MySQL INSERT | 2017-07-21 | 2017-07-22 | Start
| learning.. |
| 2 | 任务-1 | 2017-01-01 | 2017-01-02 | Descri
|ption 1 |
| 3 | 任务-2 | 2017-01-01 | 2017-01-02 | Descri
|ption 2 |
| 4 | 任务-3 | 2017-01-01 | 2017-01-02 | Descri
|ption 3 |
+-----+-----+-----+-----+
4 rows in set
```

4. MySQL INSERT与ON DUPLICATE KEY UPDATE

如果新行违反[主键\(PRIMARY KEY\)](#)或 [UNIQUE](#) 约束，MySQL会发生错误。例如，如果执行以下语句：

```
INSERT INTO tasks(task_id,subject,start_date,end_date,description
)
VALUES (4,'Test ON DUPLICATE KEY UPDATE','2017-01-01','2017-01-0
2','Next Priority');
```

MySQL很不高兴，并向你扔来一个错误消息：

```
Error Code: 1062. Duplicate entry '4' for key 'PRIMARY' 0.016 se
c
```

9.1 INSERT 语句

因为表中的主键 task_id 列已经有一个值为 4 的行了，所以该语句违反了 PRIMARY KEY 约束。

但是，如果在 INSERT 语句中指定 ON DUPLICATE KEY UPDATE 选项，MySQL 将插入新行或使用新值更新原行记录。

例如，以下语句使用新的 task_id 和 subject 来更新 task_id 为 4 的行。

```
INSERT INTO tasks(task_id,subject,start_date,end_date,description)
)
VALUES (4,'Test ON DUPLICATE KEY UPDATE','2017-01-01','2017-01-0
2','Next Priority')
ON DUPLICATE KEY UPDATE
task_id = task_id + 1,
subject = 'Test ON DUPLICATE KEY UPDATE';
```

执行上面语句后，MySQL发出消息说 2 行受影响。现在，我们来看看 tasks 表中的数据：

```
mysql> select * from tasks;
+-----+-----+-----+-----+
| task_id | subject | start_date | end_date |
| description |
+-----+-----+-----+-----+
| 1 | Learn MySQL INSERT | 2017-07-21 | 2017-07-
22 | Start learning.. |
| 2 | 任务-1 | 2017-01-01 | 2017-01-02 |
| Description 1 |
| 3 | 任务-2 | 2017-01-01 | 2017-01-02 |
| Description 2 |
| 5 | Test ON DUPLICATE KEY UPDATE | 2017-01-01 | 2017-01-
02 | Description 3 |
+-----+-----+-----+-----+
4 rows in set
```

新行没有被插入，但是更新了 task_id 值为 4 的行。上面的 INSERT ON DUPLICATE KEY UPDATE 语句等效于以下UPDATE语句：

```
UPDATE tasks
SET
    task_id = task_id + 1,
    subject = 'Test ON DUPLICATE KEY UPDATE'
WHERE
    task_id = 4;
```

有关 INSERT ON DUPLICATE KEY UPDATE 语句的更多信息，请查看[MySQL插入或更新教程](#)。

在本教程中，我们向您展示了如何使用各种形式的MySQL INSERT 语句将数据插入到表中。

在本教程中，您将学习如何使用MySQL `INSERT IGNORE` 语句将数据插入到表中。

1. MySQL INSERT IGNORE语句简介

当使用`INSERT`语句向表中添加一些行数据并且在处理期间发生错误时，`INSERT`语句将被中止，并返回错误消息。因此，可能不会向表中插入任何行。

但是，如果使用`INSERT IGNORE`语句，则会忽略导致错误的行，并将其余行插入到表中。

`INSERT IGNORE`语句的语法如下：

```
INSERT IGNORE INTO table(column_list)
VALUES( value_list),
      ( value_list),
      ...
```

请注意，`IGNORE`子句是MySQL对SQL标准的扩展。

2. MySQL INSERT IGNORE示例

为了演示，我们将创建一个名为订阅者(`subscribers`)的新表。

```
USE testdb;

CREATE TABLE IF NOT EXISTS subscribers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(50) NOT NULL UNIQUE
);
```

`UNIQUE`约束确保电子邮件列中不存在重复的电子邮件。

以下语句在`subscribers`表中插入一个新行：

```
INSERT INTO subscribers(email)
VALUES('yiibai.com@gmail.com');
```

上面语句它按预期那样工作。接下来，我们来执行另一个语句，将两行插入到 `subscribers` 表中：

```
INSERT INTO subscribers(email)
VALUES('yiibai.com@gmail.com'),
      ('jane.max@ibm.com');
```

它将返回一个错误：

```
1062 - Duplicate entry 'yiibai.com@gmail.com' for key 'email'
```

如错误消息中所示，电子邮件 `yiibai.com@gmail.com` 值重复而导致违反 `UNIQUE` 约束。

但是，如果您使用 `INSERT IGNORE` 语句，则其它语句将会继续执行 -

```
INSERT IGNORE INTO subscribers(email)
VALUES('yiibai.com@gmail.com'),
      ('jane.max@ibm.com');
```

MySQL服务器返回一条消息，显示插入一行，另一行被忽略。

```
Query OK, 1 row affected
Records: 2 Duplicates: 1 Warnings: 1
```

如果使用 `SHOW WARNINGS` 语句，就会发现警告的详细信息：

```
mysql> SHOW WARNINGS;
+-----+-----+
| Level | Code | Message
+-----+-----+
| Warning | 1062 | Duplicate entry 'yiibai.com@gmail.com' for key
  'email' |
+-----+
2 rows in set
```

所以当使用 `INSERT IGNORE` 语句来执行插入数据时，MySQL只发出警告而不是发出错误，以防发生错误退出其它数据无法插入。

如果查询 `subscribers` 表中的数据，就会发现实际只有一行数据插入，导致错误的行被忽略。

```
mysql> select * from subscribers;
+-----+
| id | email
+-----+
| 4  | jane.max@ibm.com
| 1  | yiibai.com@gmail.com
+-----+
2 rows in set
```

3. MySQL INSERT IGNORE和STRICT模式

当 `STRICT` 模式打开时，如果您尝试将无效值插入到表中，MySQL将返回错误并中止 `INSERT` 语句。

但是，如果使用 `INSERT IGNORE` 语句，则MySQL将发出警告而不是错误。此外，它将尝试调整值以使其在插入表之前有效。

请来看看以下示例。

9.2 INSERT IGNORE

首先，我们创建一个名为 `tokens` 的新表：

```
USE yiibaidb;

CREATE TABLE IF NOT EXISTS tokens (
    s VARCHAR(6)
);
```

在此表中，列只接受长度小于或等于 6 的字符串。接下来，如果将长度为 7 的字符串插入到 `tokens` 表中，那么会发生什么？

```
INSERT INTO tokens VALUES('abcdefg');
```

由于 `STRICT` 模式已打开，MySQL发出以下错误 -

```
mysql> INSERT INTO tokens VALUES('abcdefg');
1406 - Data too long for column 's' at row 1
```

现在，使用 `INSERT IGNORE` 语句重新插入相同的字符串。

```
INSERT IGNORE INTO tokens VALUES('abcdefg');
```

将MySQL截断的数据插入到 `tokens` 表中。此外，它发出警告。使用 `SHOW WARNINGS` 查看错误提示 -

```
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 's' at row 1 |
+-----+-----+-----+
1 row in set
```

在本教程中，您学习了如何使用MySQL `INSERT IGNORE` 语句将行插入到表中，并忽略导致错误的行的错误。

更新数据是使用数据库时最重要的任务之一。在本教程中，您将学习如何使用 MySQL `UPDATE` 语句来更新表中的数据。

1. MySQL UPDATE语句简介

我们使用 `UPDATE` 语句来更新表中的现有数据。也可以使用 `UPDATE` 语句来更改表中单个行，一组行或所有行的列值。

下面说明了MySQL `UPDATE` 语句的语法：

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
    column_name1 = expr1,
    column_name2 = expr2,
    ...
WHERE
    condition;
```

在上面 `UPDATE` 语句中：

- 首先，在 `UPDATE` 关键字后面指定要更新数据的表名。
- 其次，`SET` 子句指定要修改的列和新值。要更新多个列，请使用以逗号分隔的列表。以字面值，表达式或子查询的形式在每列的赋值中来提供要设置的值。
- 第三，使用`WHERE子句`中的条件指定要更新的行。`WHERE` 子句是可选的。如果省略 `WHERE` 子句，则 `UPDATE` 语句将更新表中的所有行。

请注意，`WHERE` 子句非常重要，所以不应该忘记指定更新的条件。有时，您可能只想改变一行；但是，可能会忘记写上 `WHERE` 子句，导致意外更新表中的所有行。

MySQL在 `UPDATE` 语句中支持两个修饰符。

- `LOW_PRIORITY` 修饰符指示 `UPDATE` 语句延迟更新，直到没有从表中读取数据的连接。`LOW_PRIORITY` 对仅使用表级锁定的存储引擎(例如 `MyISAM`, `MERGE`, `MEMORY`)生效。
- 即使发生错误，`IGNORE`修饰符也可以使`UPDATE`语句继续更新行。导致错误(如重复键冲突)的行不会更新。

2. MySQL UPDATE示例

我们使用MySQL示例数据库(yiibaidb)中的一些表来练习使用 UPDATE 语句。

2.1 MySQL UPDATE一个单列示例

在这个例子中，我们将把 *Mary Patterson* 的电子邮件更新为新的电子邮件 `mary.patterso@yiibai.com`。

首先，为了确保更新电子邮件成功，使用以下SELECT语句从 employees 表查询 Mary 的电子邮件：

```
SELECT
    firstname, lastname, email
FROM
    employees
WHERE
    employeeNumber = 1056;
```

执行上面的查询语句，得到以下结果 -

firstname	lastname	email
Mary	Patterson	mpatterso@yiibai.com

1 row in set

第二步，使用 UPDATE 语句将 Mary 的电子邮件更新为新的电子邮件： `mary.new@yiibai.com`，如下查询所示：

```
UPDATE employees
SET
    email = 'mary.new@yiibai.com'
WHERE
    employeeNumber = 1056;
```

9.3 UPDATE 语句

因为上面语句中，只想更新一行，所以使用 WHERE 子句来指定更新的是员工编号 1056 的行。 SET 子句将电子邮件列的值设置为新的电子邮件。

第三，再次执行 SELECT 语句来验证更改。

```
SELECT
    firstname, lastname, email
FROM
    employees
WHERE
    employeeNumber = 1056;
```

再次执行上面的查询语句，得到以下结果 -

```
+-----+-----+-----+
| firstname | lastname | email
+-----+-----+-----+
| Mary      | Patterson | mary.new@yiibai.com |
+-----+-----+-----+
1 row in set
```

2.2 MySQL UPDATE 多列

要更新多列中的值，需要在 SET 子句中指定分配。例如，以下语句更新了员工编号 1056 的姓氏和电子邮件：

```
UPDATE employees
SET
    lastname = 'Hill',
    email = 'mary.hill@yiibai.com'
WHERE
    employeeNumber = 1056;
```

在执行上面语句之后，查询员工编号为： 1056 的记录，如下所示 -

```
+-----+-----+-----+
| firstname | lastname | email      |
+-----+-----+-----+
| Mary     | Hill    | mary.hill@yiibai.com |
+-----+-----+-----+
1 row in set
```

2.3 使用 SELECT 语句的 MySQL UPDATE 示例

可以使用 `SELECT` 语句查询来自其他表的数据来提供给 `SET` 子句的值。

例如，在 `customers` 表中，有些客户没有任何销售代表。

`salesRepEmployeeNumber` 列的值为 `NULL`，如下所示：

```
mysql> SELECT
    customername, salesRepEmployeeNumber
  FROM
    customers
 WHERE
    salesRepEmployeeNumber IS NULL;
+-----+-----+
| customername | salesRepEmployeeNumber |
+-----+-----+
| Havel & Zbyszek Co | NULL |
| Porto Imports Co. | NULL |
| Asian Shopping Network, Co | NULL |
| Natrlich Autos | NULL |
| ANG Resellers | NULL |
| Messner Shopping Network | NULL |
| Franken Gifts, Co | NULL |
| BG&E Collectables | NULL |
| Schuyler Imports | NULL |
| Der Hund Imports | NULL |
| Cramer Spezialitten, Ltd | NULL |
| Asian Treasures, Inc. | NULL |
| SAR Distributors, Co | NULL |
| Kommission Auto | NULL |
| Lisboa Souveniers, Inc | NULL |
| Stuttgart Collectable Exchange | NULL |
| Feuer Online Stores, Inc | NULL |
| Warburg Exchange | NULL |
| Anton Designs, Ltd. | NULL |
| Mit Vergngen & Co. | NULL |
| Kremlin Collectables, Co. | NULL |
| Raanan Stores, Inc | NULL |
+-----+-----+
22 rows in set
```

我们可以为这些客户提供销售代表和更新。

为此，需要从 `employees` 表中随机选择一个职位为 `Sales Rep` 的雇员，并将其更新到 `employees` 表中。下面的查询语句是从 `employees` 表中随机选择一个其职位是 `Sales Rep` 的员工。

```

SELECT
    employeeNumber
FROM
    employees
WHERE
    jobtitle = 'Sales Rep'
ORDER BY RAND()
LIMIT 1;

```

要更新 `customers` 表中的销售代表员工编号(`employeeNumber`)列，我们将上面的查询放在 `UPDATE` 语句的 `SET` 子句中，如下所示：

```

UPDATE customers
SET
    salesRepEmployeeNumber = (SELECT
                                employeeNumber
                                FROM
                                employees
                                WHERE
                                jobtitle = 'Sales Rep'
                                LIMIT 1)
WHERE
    salesRepEmployeeNumber IS NULL;

```

如果在执行上面更新语句后，查询 `customers` 表中的数据，将看到每个客户都有一个销售代表。换句话说，以下查询不返回任何行数据。

```

SELECT
    salesRepEmployeeNumber
FROM
    customers
WHERE
    salesRepEmployeeNumber IS NULL;

```

在本教程中，您已经学会了如何使用MySQL `UPDATE` 语句来更新数据库表中的数据。

在本教程中，您将学习如何使用MySQL `UPDATE JOIN` 语句来执行跨表更新。我们将逐步介绍如何使用 `INNER JOIN` 子句和 `LEFT JOIN` 子句与 `UPDATE` 语句一起使用。

1. MySQL UPDATE JOIN语法

我们经常使用 `join` 子句来查询表中的行(在`INNER JOIN`的情况下)，或者可能没有(在`LEFT JOIN`的情况下)另一个表中的相应行。在MySQL中，可以在`UPDATE语句`中使用 `JOIN` 子句执行跨表更新。

MySQL `UPDATE JOIN` 的语法如下：

```
UPDATE T1, T2,
[INNER JOIN | LEFT JOIN] T1 ON T1.C1 = T2. C1
SET T1.C2 = T2.C2,
    T2.C3 = expr
WHERE condition
```

让我们更详细地看看MySQL `UPDATE JOIN` 语法：

- 首先，在 `UPDATE` 子句之后，指定主表(`T1`)和希望主表连接表(`T2`)。请注意，必须在 `UPDATE` 子句之后至少指定一个表。`UPDATE` 子句后未指定的表中的数据未更新。
- 第二，指定一种要使用的连接，即 `INNER JOIN` 或 `LEFT JOIN` 和连接条件。`JOIN` 子句必须出现在 `UPDATE` 子句之后。
- 第三，要为要更新的 `T1` 和/或 `T2` 表中的列分配新值。
- 第四，`WHERE子句`中的条件用于指定要更新的行。

如果您学习过了`UPDATE语句`教程，您可能会注意到使用以下语法更新数据交叉表的另一种方法：

```
UPDATE T1, T2
SET T1.c2 = T2.c2,
    T2.c3 = expr
WHERE T1.c1 = T2.c1 AND condition
```

在这个 `UPDATE` 语句与具有隐式 `INNER JOIN` 子句的 `UPDATE JOIN` 工作相同。这意味着可以如下重写上述语句：

```
UPDATE T1, T2  
INNER JOIN T2 ON T1.C1 = T2.C1  
SET T1.C2 = T2.C2,  
    T2.C3 = expr  
WHERE condition
```

让我们来看一些使用 `UPDATE JOIN` 语句来更好地理解的例子。

2. MySQL UPDATE JOIN示例

我们将在这些例子中使用一个新的示例数据库(`empdb`)。示例数据库包含 2 个表：

- `employees` 表将存储在员工编号，姓名，工作表现和工资的数据。
- `merits` 表存储员工绩效和绩效百分比。

以下语句在 `empdb` 示例数据库中创建表并导入数据：

```

CREATE DATABASE IF NOT EXISTS empdb;

USE empdb;
-- create tables
CREATE TABLE merits (
    performance INT(11) NOT NULL,
    percentage FLOAT NOT NULL,
    PRIMARY KEY (performance)
);

CREATE TABLE employees (
    emp_id INT(11) NOT NULL AUTO_INCREMENT,
    emp_name VARCHAR(255) NOT NULL,
    performance INT(11) DEFAULT NULL,
    salary FLOAT DEFAULT NULL,
    PRIMARY KEY (emp_id),
    CONSTRAINT fk_performance FOREIGN KEY (performance)
        REFERENCES merits (performance)
);
-- insert data for merits table
INSERT INTO merits(performance,percentage)
VALUES(1,0),
      (2,0.01),
      (3,0.03),
      (4,0.05),
      (5,0.08);
-- insert data for employees table
INSERT INTO employees(emp_name,performance,salary)
VALUES('Mary Doe', 1, 50000),
      ('Cindy Minsu', 3, 65000),
      ('Sue Greenspan', 4, 75000),
      ('Grace Dell', 5, 125000),
      ('Nancy Johnson', 3, 85000),
      ('John Doe', 2, 45000),
      ('Lily Bush', 3, 55000);

```

2.1 使用INNER JOIN子句的MySQL UPDATE JOIN示例

假设想根据员工的工作表现来调整员工的工资。

因此，优点百分比存储在 `merits` 表中，您必须使用 `UPDATE INNER JOIN` 语句根据存储在 `merits` 表中的百分比来调整 `employees` 表中员工的工资。

`employees` 和 `merits` 表之间以是 `performance` 字段相关联的。请参阅以下查询：

```
UPDATE employees
    INNER JOIN
        merits ON employees.performance = merits.performance
SET
    salary = salary + salary * percentage;
```

上面查询语句的工作原理是什么？

我们仅在 `UPDATE` 子句之后指定 `employees` 表，因为我们只想更新 `employees` 表中的数据。

对于 `employees` 表中的每一行，查询根据 `merits` 表中 `performance` 列中的值来检查 `employees` 表中的 `performance` 列中的值。如果找到一个匹配，它将获得 `merits` 表中的百分比，并更新 `employees` 表中的 `salary` 列。

```

mysql> select * from employees; -- 更新之前的数据
+-----+-----+-----+-----+
| emp_id | emp_name | performance | salary |
+-----+-----+-----+-----+
| 1     | Mary Doe   | 1           | 50000  |
| 2     | Cindy Minsu | 3           | 65000  |
| 3     | Sue Greenspan | 4           | 75000  |
| 4     | Grace Dell  | 5           | 125000 |
| 5     | Nancy Johnson | 3           | 85000  |
| 6     | John Doe    | 2           | 45000  |
| 7     | Lily Bush   | 3           | 55000  |
+-----+-----+-----+-----+
7 rows in set

mysql> UPDATE employees
      INNER JOIN
          merits ON employees.performance = merits.performance
      SET
          salary = salary + salary * percentage; -- 执行连接更新
Query OK, 6 rows affected
Rows matched: 7  Changed: 6  Warnings: 0

mysql> select * from employees; -- 更新之后的数据
+-----+-----+-----+-----+
| emp_id | emp_name | performance | salary |
+-----+-----+-----+-----+
| 1     | Mary Doe   | 1           | 50000  |
| 2     | Cindy Minsu | 3           | 66950  |
| 3     | Sue Greenspan | 4           | 78750  |
| 4     | Grace Dell  | 5           | 135000 |
| 5     | Nancy Johnson | 3           | 87550  |
| 6     | John Doe    | 2           | 45450  |
| 7     | Lily Bush   | 3           | 56650  |
+-----+-----+-----+-----+
7 rows in set

```

因为省略了 `UPDATE` 语句中的 `WHERE` 子句，所以 `employees` 表中的所有记录都被更新。

2.2 具有LEFT JOIN的MySQL UPDATE JOIN示例

9.4 UPDATE JOIN 语句

假设公司又雇用了两名新员工：

```
INSERT INTO employees(emp_name, performance, salary)
VALUES('Jack William', NULL, 43000),
      ('Ricky Bond', NULL, 52000);
```

因为这些员工是新员工，所以他们的绩效(`performance`)数据不可用或为 `NULL`。现在 `employees` 表中的数据，如下所示 -

```
mysql> SELECT * FROM employees;
+-----+-----+-----+-----+
| emp_id | emp_name | performance | salary |
+-----+-----+-----+-----+
| 1     | Mary Doe   | 1           | 50000  |
| 2     | Cindy Minsu | 3           | 66950  |
| 3     | Sue Greenspan | 4           | 78750  |
| 4     | Grace Dell  | 5           | 135000 |
| 5     | Nancy Johnson | 3           | 87550  |
| 6     | John Doe    | 2           | 45450  |
| 7     | Lily Bush   | 3           | 56650  |
| 8     | Jack William | NULL        | 43000  |
| 9     | Ricky Bond  | NULL        | 52000  |
+-----+-----+-----+-----+
9 rows in set
```

要计算新员工的工资，不能使用 `UPDATE INNER JOIN` 语句，因为它们的绩效数据在 `merits` 表中不可用。这就是为什么要使用 `UPDATE LEFT JOIN` 来实现了。

当 `UPDATE LEFT JOIN` 语句在另一个表中没有相应行时，就会更新表中的一行。

例如，您可以使用以下语句将新雇员的工资增加 1.5%：

```
UPDATE employees
  LEFT JOIN
    merits ON employees.performance = merits.performance
SET
  salary = salary + salary * 0.015
WHERE
  merits.percentage IS NULL;
```

上面语句执行后，得到以下结果 -

```
mysql> SELECT * FROM employees;
+-----+-----+-----+-----+
| emp_id | emp_name | performance | salary |
+-----+-----+-----+-----+
| 1     | Mary Doe   | 1           | 50000  |
| 2     | Cindy Minsu | 3           | 66950  |
| 3     | Sue Greenspan | 4           | 78750  |
| 4     | Grace Dell  | 5           | 135000 |
| 5     | Nancy Johnson | 3           | 87550  |
| 6     | John Doe    | 2           | 45450  |
| 7     | Lily Bush   | 3           | 56650  |
| 8     | Jack William | NULL        | 43000  |
| 9     | Ricky Bond  | NULL        | 52000  |
+-----+-----+-----+-----+
9 rows in set

mysql> UPDATE employees
      LEFT JOIN
          merits ON employees.performance = merits.performance
      SET
          salary = salary + salary * 0.015
      WHERE
          merits.percentage IS NULL;
Query OK, 2 rows affected
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from employees;
+-----+-----+-----+-----+
| emp_id | emp_name | performance | salary |
+-----+-----+-----+-----+
| 1     | Mary Doe   | 1           | 50000  |
| 2     | Cindy Minsu | 3           | 66950  |
| 3     | Sue Greenspan | 4           | 78750  |
| 4     | Grace Dell  | 5           | 135000 |
| 5     | Nancy Johnson | 3           | 87550  |
| 6     | John Doe    | 2           | 45450  |
| 7     | Lily Bush   | 3           | 56650  |
| 8     | Jack William | NULL        | 43645  |
| 9     | Ricky Bond  | NULL        | 52780  |
+-----+-----+-----+-----+
```

9.4 UPDATE JOIN 语句

```
+-----+-----+-----+-----+
9 rows in set
```

在本教程中，我们向您展示了如何使用MySQL UPDATE JOIN 与 INNER JOIN 和 LEFT JOIN 子句来执行跨表更新。

在本教程中，您将学习如何使用MySQL `DELETE` 语句从单个表中删除数据。

1. MySQL `DELETE`语句介绍

要从表中删除数据，请使用MySQL `DELETE` 语句。下面说明了 `DELETE` 语句的语法：

```
DELETE FROM table_name  
WHERE condition;
```

在上面查询语句中 -

- 首先，指定删除数据的表(`table_name`)。
- 其次，使用条件来指定要在 `WHERE` 子句中删除的行记录。如果行匹配条件，这些行记录将被删除。

请注意，`WHERE` 子句是可选的。如果省略 `WHERE` 子句，`DELETE` 语句将删除表中的所有行。

除了从表中删除数据外，`DELETE` 语句返回删除的行数。

要使用单个 `DELETE` 语句从多个表中删除数据，请阅读下一个教程中将介绍的 [DELETE JOIN语句](#)。

要删除表中的所有行，而不需要知道删除了多少行，那么应该使用[TRUNCATE TABLE](#)语句来获得更好的执行性能。

对于具有[外键约束](#)的表，当从父表中删除行记录时，子表中的行记录将通过使用[ON DELETE CASCADE](#)选项自动删除。

2. MySQL `DELETE`的例子

我们将使用[示例数据库\(yiibaidb\)](#)中的 `employees` 表进行演示。

Field	Type	Null	Key	Default	Extra
emp_id	int(11)	NO	PRI	NULL	auto_increment
emp_name	varchar(255)	NO		NULL	
performance	int(11)	YES	MUL	NULL	
salary	float	YES		NULL	

4 rows in set

请注意，一旦删除数据，它就会永远消失。因此，在执行 `DELETE` 语句之前，应该先 **备份数据库**，以防万一要找回删除过的数据。

假设要删除 `officeNumber` 为 4 的员工，则使用 `DELETE` 语句与 `WHERE` 子句作为以下查询：

```
DELETE FROM employees
WHERE
    officeCode = 4;
```

要删除 `employees` 表中的所有行，请使用不带 `WHERE` 子句的 `DELETE` 语句，如下所示：

```
DELETE FROM employees;
```

在执行上面查询语句后，`employees` 表中的所有行都被删除。

MySQL DELETE和LIMIT子句

如果要限制要删除的行数，则使用 `LIMIT` 子句，如下所示：

```
DELETE FROM table  
LIMIT row_count;
```

请注意，表中的行顺序未指定，因此，当您使用 `LIMIT` 子句时，应始终使用 `ORDER BY` 子句，不然删除的记录可能不是你所预期的那样。

```
DELETE FROM table_name  
ORDER BY c1, c2, ...  
LIMIT row_count;
```

考虑在 [示例数据库\(yiibaidb\)](#) 中的 `customers` 表，其表结构如下：

```
mysql> desc customers;
+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default |
| Extra          |            |       |      |          |
+-----+-----+-----+-----+
| customerNumber | int(11)    | NO   | PRI | NULL    |
| customerName   | varchar(50) | NO   |      | NULL    |
| contactLastName | varchar(50) | NO   |      | NULL    |
| contactFirstName | varchar(50) | NO   |      | NULL    |
| phone          | varchar(50) | NO   |      | NULL    |
| addressLine1   | varchar(50) | NO   |      | NULL    |
| addressLine2   | varchar(50) | YES  |      | NULL    |
| city           | varchar(50) | NO   |      | NULL    |
| state          | varchar(50) | YES  |      | NULL    |
| postalCode     | varchar(15) | YES  |      | NULL    |
| country         | varchar(50) | NO   |      | NULL    |
| salesRepEmployeeNumber | int(11)    | YES  | MUL | NULL    |
| creditLimit    | decimal(10,2) | YES  |      | NULL    |
+-----+-----+-----+-----+
13 rows in set
```

例如，以下语句按客户名称按字母排序客户，并删除前 10 个客户：

```
DELETE FROM customers  
ORDER BY customerName  
LIMIT 10;
```

类似地，以下 `DELETE` 语句选择法国(*France*)的客户，按升序按信用额度 (`creditLimit`)进行排序，并删除前 5 个客户：

```
DELETE FROM customers  
WHERE country = 'France'  
ORDER BY creditLimit  
LIMIT 5;
```

类似地，以下 `DELETE` 语句选择法国(*France*)的客户，按升序按信用额度 (`creditLimit`)进行排序，并删除前 5 个客户：

```
DELETE FROM customers  
WHERE country = 'France'  
ORDER BY creditLimit  
LIMIT 5;
```

在本教程中，您已经学会了如何使用MySQL `DELETE` 语句从表中删除数据。

在本教程中，您将学习如何使用MySQL `ON DELETE CASCADE` 引用操作来执行外键从多个相关表中删除数据。

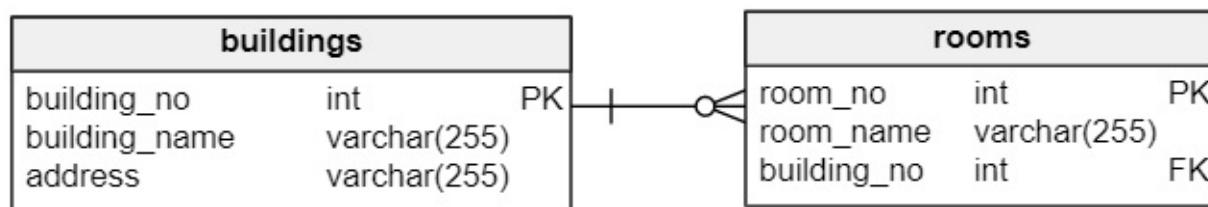
在上一个教程中，我们学习了如何使用单个`DELETE`语句从一个或多个相关表中删除数据。但是，MySQL提供了一种更为有效的方法，称为 `ON DELETE CASCADE` 对于外键的引用操作，可以实现在从父表中删除数据时自动删除子表中的数据。

1. MySQL ON DELETE CASCADE示例

下面来看一些使用MySQL `ON DELETE CASCADE` 的例子。

假设有两张表：建筑物(`buildings`)和房间(`rooms`)。在这个数据库模型中，每个建筑物都有一个或多个房间。然而，每个房间只属于一个建筑物。没有建筑物则房间是不会存在的。

建筑物和房间表之间的关系是一对多($1:N$)，如下面的数据库图所示：



当我们从 `buildings` 表中删除一行时，还要删除 `rooms` 表中引用建筑物表中行的行。例如，当删除建筑编号(`building_no`)为 2 的行记录时，在 `buildings` 表上执行如下查询：

```
DELETE FROM buildings
WHERE
    building_no = 2;
```

我们希望 `rooms` 表中涉及到建筑物编号 2 的行记录也将被删除(讲得通俗一点：假设 2 号楼倒塌了，那么 2 号楼的房间应该也就不存在了)。以下是演示MySQL `ON DELETE CASCADE` 参考操作如何工作的步骤。

第一步，创建 `buildings` 表，如下创建语句：

```
USE testdb;
CREATE TABLE buildings (
    building_no INT PRIMARY KEY AUTO_INCREMENT,
    building_name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

第二步，创建 rooms 表，如下创建语句：

```
USE testdb;
CREATE TABLE rooms (
    room_no INT PRIMARY KEY AUTO_INCREMENT,
    room_name VARCHAR(255) NOT NULL,
    building_no INT NOT NULL,
    FOREIGN KEY (building_no)
        REFERENCES buildings (building_no)
        ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

请注意，在外键约束定义的末尾添加 ON DELETE CASCADE 子句。

第三步，将一些数据插入到 buildings 表，如下插入语句：

```
INSERT INTO buildings(building_name, address)
VALUES('海南大厦', '海口市国兴大道1234号'),
      ('万达水城', '海口市大同路1200号');
```

第四步，查询 buildings 表中的数据：

```
mysql> select * from buildings;
+-----+-----+
| building_no | building_name | address |
+-----+-----+
| 1 | 海南大厦 | 海口市国兴大道1234号 |
| 2 | 万达水城 | 海口市大同路1200号 |
+-----+-----+
2 rows in set
```

现在可以看到，在建筑物表中有两行记录。

第五步，将一些数据插入到 `buildings` 表，如下插入语句：

```
INSERT INTO rooms(room_name,building_no)
VALUES('Amazon',1),
      ('War Room',1),
      ('Office of CEO',1),
      ('Marketing',2),
      ('Showroom',2);
```

第六步，查询 `rooms` 表中的数据：

```
mysql> select * from rooms;
+-----+-----+-----+
| room_no | room_name | building_no |
+-----+-----+-----+
| 1 | Amazon | 1 |
| 2 | War Room | 1 |
| 3 | Office of CEO | 1 |
| 4 | Marketing | 2 |
| 5 | Showroom | 2 |
+-----+-----+-----+
5 rows in set
```

从上面行记录中可以看到，`building_no=1`的建筑有 3 个房间，以及`building_no=2`有 2 个房间。

第七步，删除编号为2的建筑物：

```
DELETE FROM buildings WHERE building_no = 2;
```

第八步，查询 `rooms` 表中的数据 -

```
mysql> DELETE FROM buildings WHERE building_no = 2;
Query OK, 1 row affected

mysql> SELECT * FROM rooms;
+-----+-----+-----+
| room_no | room_name | building_no |
+-----+-----+-----+
| 1      | Amazon    | 1          |
| 2      | War Room   | 1          |
| 3      | Office of CEO | 1          |
+-----+-----+-----+
3 rows in set
```

可以看到，表中只剩下引用 `building_no=1` 的记录了，引用 `building_no=2` 的所有行记录都被自动删除了。

请注意，`ON DELETE CASCADE` 仅支持使用存储引擎支持外键(如 InnoDB)的表上工作。某些表类型不支持诸如 MyISAM 的外键，因此应该在使用MySQL `ON DELETE CASCADE` 引用操作的表上选择适当的存储引擎。

查找受MySQL ON DELETE CASCADE操作影响的表的技巧

有时，当要从表中删除数据时，知道哪个表受到MySQL `ON DELETE CASCADE` 参考操作的影响是有用的。可从 `information_schema` 数据库中的 `referential_constraints` 表中查询此数据，如下所示：

```
USE information_schema;

SELECT
    table_name
FROM
    referential_constraints
WHERE
    constraint_schema = 'database_name'
        AND referenced_table_name = 'parent_table'
        AND delete_rule = 'CASCADE'
```

例如，要使用示例数据库(`testdb`，因为上面两个表是建立在 `testdb` 数据库之上的)中的 `CASCADE` 删除规则查找与建筑表相关联的表，请使用以下查询：

```
USE information_schema;

SELECT
    table_name
FROM
    referential_constraints
WHERE
    constraint_schema = 'testdb'
        AND referenced_table_name = 'buildings'
        AND delete_rule = 'CASCADE'
```

执行上面查询语句，得到以下结果 -

```
+-----+
| table_name |
+-----+
| rooms      |
+-----+
1 row in set
```

在本教程中，我们一步一步向您展示了如何在从父表中删除数据时，使用MySQL
ON DELETE CASCADE 引用操作从外键自动从子表中删除相关联的数据。

在本教程中，我们将向您展示如何使用MySQL `DELETE JOIN` 语句来从多个表中删除数据。

在上一个教程中，我们学习了如何使用以下方式删除多个表的行记录：

- 在多个表上使用单个 `DELETE` 语句删除数据。
- 多个相关表上的使用单个 `DELETE` 语句，子表具有对外键的 `ON DELETE CASCADE` 引用操作。

本教程将介绍一种使用 `INLEER JOIN` 或 `LEFT JOIN` 子句与 `DELETE` 语句从多个表中删除数据的更灵活的方法。

MySQL DELETE语句使用INNER JOIN子句

MySQL还允许在 `DELETE` 语句中使用 `INNER JOIN` 子句来从表中删除和另一个表中的匹配的行记录。

例如，要从符合指定条件的 `T1` 和 `T2` 表中删除行记录，请使用以下语句：

```
DELETE T1, T2
FROM T1
INNER JOIN T2 ON T1.key = T2.key
WHERE condition
```

请注意，将 `T1` 和 `T2` 表放在 `DELETE` 和 `FROM` 关键字之间。如果省略 `T1` 表，`DELETE` 语句仅删除 `T2` 表中的行记录。同样，如果省略了 `T2` 表，`DELETE` 语句将只删除 `T1` 表中的行记录。

表达式 `T1.key = T2.key` 指定了将被删除的 `T1` 和 `T2` 表之间的匹配行记录的条件。

`WHERE` 子句中的条件确定 `T1` 和 `T2` 表中要被删除的行记录。

MySQL DELETE语句使用INNER JOIN示例

假设有两个表 `t1` 和 `t2`，其结构和数据如下：

```

USE testdb;

DROP TABLE IF EXISTS t1, t2;

CREATE TABLE t1 (
    id INT PRIMARY KEY AUTO_INCREMENT
);

CREATE TABLE t2 (
    id VARCHAR(20) PRIMARY KEY,
    ref INT NOT NULL
);

INSERT INTO t1 VALUES (1),(2),(3);

INSERT INTO t2(ref) VALUES('A',1),('B',2),('C',3);

```

连接删除语句如下图中所示 -

id		id	Ref
1	X	A	1
2		B	2
3		C	3

```

DELETE
    t1 , t2
FROM
    t1
    INNER JOIN t2
        ON t2.ref = t1.id
WHERE
    t1.id = 1;

```

以下语句删除 t1 表中 id=1 的行，并使用 DELETE ... INNER JOIN 语句删除 t2 表中的 ref=1 的行记录：

```
DELETE t1 , t2 FROM t1
    INNER JOIN
        t2 ON t2.ref = t1.id
WHERE
    t1.id = 1;
```

该语句返回以下消息：

```
2 row(s) affected
```

它表示一共有两行记录已被删除了。

MySQL DELETE与LEFT JOIN子句

我们经常在 `SELECT` 语句中使用 `LEFT JOIN` 子句来查找左表中以及右表中不匹配行的行记录。

我们还可以在 `DELETE` 语句中使用 `LEFT JOIN` 子句删除表(左表)中没有与其他表(右表)中的匹配的行记录。

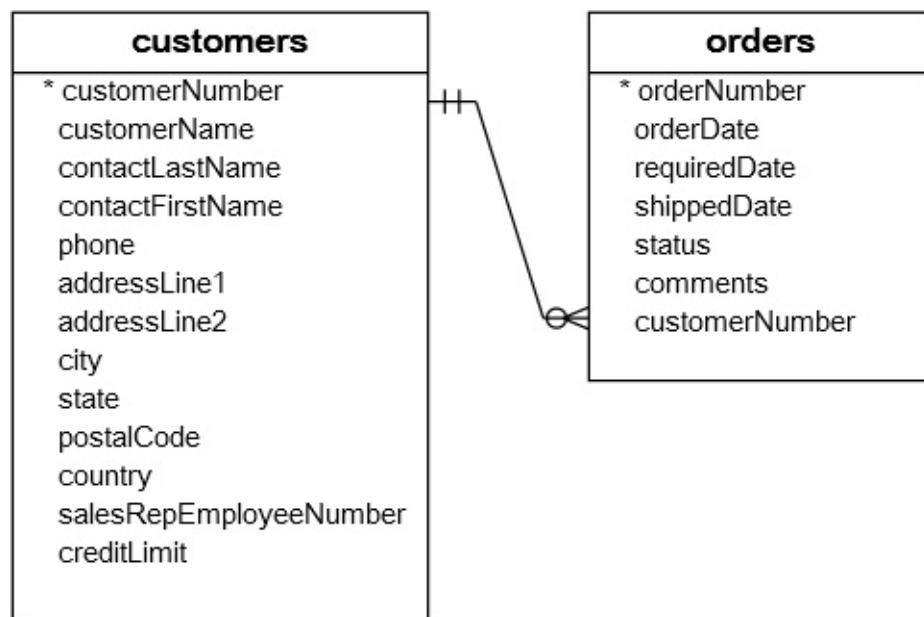
以下语法说明如何使用 `DELETE` 语句与 `LEFT JOIN` 子句来删除与 `T2` 表中没有相应匹配行的表 `T1` 中的行记录：

```
DELETE T1
FROM T1
    LEFT JOIN
        T2 ON T1.key = T2.key
WHERE
    T2.key IS NULL;
```

请注意，我们只将 `T1` 表放在 `DELETE` 关键字之后，而不是像 `INNER JOIN` 子句那样使用两个表名：`T1` 和 `T2`。

MySQL DELETE连接LEFT JOIN示例

在示例数据库(`yiibaidb`)中查看以下 `orders` 和 `orderdetails` 表：



每个客户都有零个或多个订单。但是，每个订单都属于唯一的一个客户。

可以使用 `DELETE` 语句与 `LEFT JOIN` 子句来清理客户数据。以下声明删除未下订单的客户：

```

DELETE customers
FROM customers
    LEFT JOIN
        orders ON customers.customerNumber = orders.customerNumber
WHERE
    orderNumber IS NULL;
  
```

可以通过查询没有任何订单的客户，使用以下查询来验证删除：

```

SELECT
    c.customerNumber,
    c.customerName,
    orderNumber
FROM
    customers c
    LEFT JOIN
        orders o ON c.customerNumber = o.customerNumber
WHERE
    orderNumber IS NULL;
  
```

该查询返回一个空结果集，这正如我们所预期的那样。

在本教程中，您已经学会了如何使用MySQL `DELETE JOIN` 语句从两个或多个表中删除数据。

在本教程中，您将学习如何使用MySQL REPLACE 语句向数据库表插入或更新数据。

MySQL REPLACE语句简介

MySQL REPLACE 语句是标准SQL的MySQL扩展。MySQL REPLACE 语句的工作原理如下：

- 如果给定行数据不存在，那么MySQL REPLACE语句会插入一个新行。
- 如果给定行数据存在，则 REPLACE 语句首先删除旧行，然后插入一个新行。
在某些情况下， REPLACE 语句仅更新现有行。

MySQL使用PRIMARY KEY或 UNIQUE KEY 索引来要确定表中是否存在新行。如果表没有这些索引，则 REPLACE 语句等同于INSERT语句。

要使用MySQL REPLACE 语句，至少需要具有 INSERT 和 DELETE 权限。

请注意，有一个REPLACE字符串函数，它不是本教程中所述的 REPLACE 语句。

MySQL REPLACE语句示例

我们来看一下使用 REPLACE 语句来更好地了解它的工作原理的例子。

首先，创建一个名为 cities 的新表，如下所示：

```
USE testdb;

CREATE TABLE cities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    population INT NOT NULL
);
```

接下来，在 cities 表中插入一些行记录：

```
INSERT INTO cities(name, population)
VALUES('New York', 8008278),
      ('Los Angeles', 3694825),
      ('Shanghai', 1923400);
```

下面查询 cities 表中的数据，以验证插入操作 -

```
mysql> SELECT * FROM cities;
+----+-----+-----+
| id | name        | population |
+----+-----+-----+
| 1  | New York    | 8008278   |
| 2  | Los Angeles  | 3694825   |
| 3  | Shanghai     | 1923400   |
+----+-----+-----+
3 rows in set
```

现在，在 cities 表中有三个城市数据。

那么，假设我们要将纽约市的人口更新为 1008256 ，可以使用[UPDATE语句](#)如下：

```
UPDATE cities
SET
  population = 1008256
WHERE
  id = 1;
```

再次查询 cities 表中的数据来验证更新结果 -

```

mysql> UPDATE cities
SET
    population = 1008256
WHERE
    id = 1;
Query OK, 1 row affected
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM cities;
+----+-----+-----+
| id | name      | population |
+----+-----+-----+
| 1  | New York  | 1008256   |
| 2  | Los Angeles| 3694825  |
| 3  | Shanghai   | 1923400   |
+----+-----+-----+
3 rows in set

```

UPDATE 语句的确按照预期更新了数据。

之后，使用 REPLACE 声明将洛杉矶市的人口更新为 3696820。

```

REPLACE INTO cities(id,population) VALUES(2,3696820);
Query OK, 2 rows affected

```

上面执行返回结果中提示：2 rows affected，说明有两行数据受影响。

最后，再次查询城市表的数据来验证替换的结果。

```

mysql> SELECT * FROM cities;
+----+-----+-----+
| id | name      | population |
+----+-----+-----+
| 1  | New York  | 1008256   |
| 2  | NULL       | 3696820  |
| 3  | Shanghai   | 1923400   |
+----+-----+-----+
3 rows in set

```

现在 `name` 列为 `NULL`。您可能期望 `name` 列的值保持不变。但是，`REPLACE` 语句不这样做。在这种情况下，`REPLACE` 语句的工作原理如下：

- `REPLACE` 语句首先使用列列表提供的信息将新行插入到 `cities` 表中。但是插入失败，因为在 `cities` 表中已经存在 `ID` 为 `2` 的行记录，因此，MySQL 引发了重复键错误。
- 然后，`REPLACE` 语句更新具有 `id` 列值为 `2` 指定的行记录。在正常进程中，它将先删除具有冲突 `id` 为 `2` 的旧行，然后插入一个新行。

MySQL REPLACE和INSERT

`REPLACE` 语句的第一种形式类似于 `INSERT` 语句，除了 `INSERT` 关键字换成 `REPLACE` 关键字以外，如下所示：

```
REPLACE INTO table_name(column_list)
VALUES(value_list);
```

例如，如果要在 `cities` 表中插入新行，请使用以下查询：

```
REPLACE INTO cities(name,population)
VALUES('Phoenix',1321523);
```

请注意，没有出现在 `REPLACE` 语句中的列将使用默认值插入相应的列。如果列具有 `NOT NULL` 属性并且没有默认值，并且您如果没有在 `REPLACE` 语句中指定该值，则 MySQL 将引发错误。这是 `REPLACE` 和 `INSERT` 语句之间的区别。

例如，在以下语句中，仅指定 `name` 列的值，而没有指定 `population` 列。MySQL 引发错误消息。因为 `population` 列不接受 `NULL` 值，而我们定义 `cities` 表时，也没有指定 `population` 列的默认值。

```
REPLACE INTO cities(name)
VALUES('Houston');
```

执行上面语句后，MySQL 发出如下的错误消息：

```
Error Code: 1364. Field 'population' doesn't have a default value
```



MySQL REPLACE和UPDATE

REPLACE 语句的第二种形式类似于 UPDATE 语句，如下所示：

```
REPLACE INTO table
SET column1 = value1,
    column2 = value2;
```

请注意， REPLACE 语句中没有 WHERE 子句。

例如，如果要将 Phoenix 城市的人口更新为 1768980，请使用 REPLACE 语句，如下所示：

```
REPLACE INTO cities
SET id = 4,
    name = 'Phoenix',
    population = 1768980;
```

与 UPDATE 语句不同，如果不在 SET 子句中指定列的值，则 REPLACE 语句将使用该列的默认值。

```
SELECT * FROM cities;
```

MySQL REPLACE INTO和SELECT

REPLACE 语句的第三种形式类似于 INSERT INTO SELECT 语句：

```
REPLACE INTO table_1(column_list)
SELECT column_list
FROM table_2
WHERE where_condition;
```

假设要复制 ID 为 1 的城市行记录，可以使用 REPLACE INTO SELECT 语句，如下查询示例：

```
REPLACE INTO cities(name, population)
SELECT name, population FROM cities
WHERE id = 1;
```

MySQL REPLACE语句用法

使用 REPLACE 语句时需要知道几个重点：

- 如果您开发的应用程序不仅支持MySQL数据库，而且还支持其他关系数据库管理系统(RDBMS)，则应避免使用 REPLACE 语句，因为其他RDBMS可能不支持。代替的作法是在事务中使用DELETE和INSERT语句的组合。
- 如果在具有触发器的表中使用了 REPLACE 语句，并且发生了重复键错误的删除，则触发器将按以下顺序触发：在删除前删除，删除之后，删除后，如果 REPLACE 语句删除当前行并插入新行。如果 REPLACE 语句更新当前行，则触发 BEFORE UPDATE 和 AFTER UPDATE 触发器。

在本教程中，您学习了不同形式的 REPLACE 语句来插入或更新表中的数据。

在本教程中，您将学习如何使用MySQL准备(Prepared)语句来使您的查询执行得更快更安全。

MySQL Prepared语句简介

之前的MySQL版本4.1，查询以文本格式发送到MySQL服务器。之后，MySQL服务器使用文本协议将数据返回给客户端。MySQL必须完全解析查询，并将结果集转换为字符串，然后再将其返回给客户端。

文本协议具有严重的性能问题。为了解决这个问题，MySQL自版本4.1以来添加了一个名为 `prepare` 语句的来实现一些新功能。

`prepare` 语句利用客户端/服务器二进制协议。它将包含占位符(?)的查询传递给MySQL服务器，如下例所示：

```
SELECT *
FROM products
WHERE productCode = ?;
```

当MySQL使用不同的 `productcode` 值执行此查询时，不必完全解析查询。因此，这有助于MySQL更快地执行查询，特别是当MySQL多次执行查询时。因为 `prepare` 语句使用占位符(?)，这有助于避免SQL注入的问题，从而使您的应用程序更安全一些。

MySQL准备语句用法

为了使用MySQL准备语句，您需要使用其他三个MySQL语句如下：

- ***PREPARE*** - 准备执行的声明。
- ***EXECUTE*** - 执行由 `PREPARE` 语句定义的语句。
- ***DEALLOCATE PREPARE*** - 发布 `PREPARE` 语句。

下图说明了如何使用 `PREPARE` 语句：



MySQL PREPARE语句示例

我们来看一下使用MySQL PREPARE语句的例子。

```
PREPARE stmt1 FROM 'SELECT productCode, productName  
                      FROM products  
                     WHERE productCode = ?';  
  
SET @pc = 'S10_1678';  
EXECUTE stmt1 USING @pc;  
  
DEALLOCATE PREPARE stmt1;
```

首先，使用 `PREPARE` 语句准备执行语句。我们使用 `SELECT` 语句根据指定的产品代码从 `products` 表查询产品数据。然后再使用问号(?)作为产品代码的占位符。

接下来，声明了一个产品代码变量 `@pc`，并将其值设置为 `S10_1678`。

然后，使用 `EXECUTE` 语句来执行产品代码变量 `@pc` 的准备语句。

最后，我们使用 `DEALLOCATE PREPARE` 来发布 `PREPARE` 语句。

在本教程中，我们向您展示了如何使用MySQL `PREPARE` 语句来执行带占位符的查询，以提高查询的速度，并使您的查询更安全。

在本教程中，您将了解MySQL事务以及如何使用MySQL `COMMIT` 语句和MySQL `ROLLBACK` 语句来管理MySQL中的事务。

MySQL事务介绍

要了解MySQL中的事务是什么，我们先来看看在[示例数据库\(yiibaidb\)](#)中添加新的销售订单的示例。添加销售订单的步骤如下所述：

- 从 `orders` 表中查询最新的销售订单编号，并使用下一个销售订单编号作为新的销售订单编号。
- 在指定客户的 `orders` 表中插入新的销售订单。
- 将新的销售订单项目插入 `orderdetails` 表中。
- 从 `orders` 表和 `orderdetails` 中获取数据以确认更改。

现在想象如果由于数据库故障而导致上述一个或多个步骤失败，那么数据会发什么？如果将订单项添加到 `orderdetails` 表中的步骤失败，系统中将会有空的销售订单(只有订单号，不知道这个订单卖了什么)。数据可能不完整，那么必须花费的精力来解决这个问题。

如何解决这个问题？这就是为什么事务来处理。MySQL事务使您能够执行一组MySQL操作，以确保数据库从不包含部分操作的结果。在一组操作中，如果其中一个失败，则会恢复回滚数据库。如果没有发生错误，则将整个语句集合提交到数据库。

使用MySQL事务

在上述示例中，我们将使用它们添加销售订单之前，先来看看MySQL事务语句。

要启动事务，请使用 `START TRANSACTION` 语句。要撤消MySQL语句执行，请使用 `ROLLBACK` 语句。

请注意，有一些SQL语句，主要是数据定义语句，不能在事务中使用以下语句：

```
CREATE / ALTER / DROP DATABASE  
CREATE / ALTER / DROP / RENAME / TRUNCATE TABLE  
CREATE / DROP INDEX  
CREATE / DROP EVENT  
CREATE / DROP FUNCTION  
CREATE / DROP PROCEDURE  
...
```

要将更改写入事务中的数据库，请使用 `COMMIT` 语句。要注意的是，默认情况下，MySQL 自动提交对数据库的更改。

要强制 MySQL 不会自动提交更改，请使用以下语句：

```
SET autocommit = 0
```

MySQL 事务的例子

要使用 MySQL 事务，首先必须将 MySQL 语句分解成逻辑部分，并确定何时应该提交或回滚数据。

下面来看一下使用 MySQL 事务在上面的示例数据库中添加新的销售订单并添加事务处理步骤的例子：

- 使用 `START TRANSACTION` 语句启动事务。
- 从 `orders` 表中获取最新的销售订单编号，并使用下一个销售订单编号作为新的销售订单编号。
- 在指定 `orders` 表中插入新的销售订单。
- 将新的销售订单项目插入 `orderdetails` 表中。
- 使用 `COMMIT` 语句提交更改。
- 从 `orders` 表和 `orderdetails` 表中获取数据以确认更改。

以下是执行上述步骤的脚本：

```

-- start a new transaction
start transaction;

-- get latest order number
select @orderNumber := max(orderNUmber)
from orders;
-- set new order number
set @orderNumber = @orderNumber + 1;

-- insert a new order for customer 145
insert into orders(orderNumber,
                    orderDate,
                    requiredDate,
                    shippedDate,
                    status,
                    customerNumber)
values(@orderNumber,
       now(),
       date_add(now(), INTERVAL 5 DAY),
       date_add(now(), INTERVAL 2 DAY),
       'In Process',
       145);

-- insert 2 order line items
insert into orderdetails(orderNumber,
                         productCode,
                         quantityOrdered,
                         priceEach,
                         orderLineNumber)
values(@orderNumber, 'S18_1749', 30, '136', 1),
      (@orderNumber, 'S18_2248', 50, '55.09', 2);

-- commit changes
commit;

-- get the new inserted order
select * from orders a
inner join orderdetails b on a.ordernumber = b.ordernumber
where a.ordernumber = @ordernumber;

```

在本教程中，您学习了如何使用 `START TRANSACTION COMMIT` 和 `ROLLBACK` 的 MySQL 事务语句来管理 MySQL 中的事务以保护数据完整性。

在本教程中，您将学习如何使用MySQL锁来协调会话之间的表访问。

MySQL允许客户端会话明确获取表锁，以防止其他会话在特定时间段内访问表。客户端会话只能为自己获取或释放表锁。它不能获取或释放其他会话的表锁。

在详细介绍之前，我们将创建一个名为 `sampledb` 的示例数据库，其中包含一个简单的 `tbl` 表来模拟练习表锁定语句。

```
CREATE DATABASE IF NOT EXISTS testdb;

USE testdb;
CREATE TABLE tbl (
    id int(11) NOT NULL AUTO_INCREMENT,
    col int(11) NOT NULL,
    PRIMARY KEY (id)
);
```

LOCK和UNLOCK TABLES语法

获取表的锁的简单形式如下：

```
LOCK TABLES table_name [READ | WRITE]
```

可将表的名称放在 `LOCK TABLES` 关键字后面，后跟一个锁类型。MySQL提供两种锁类型：`READ` 和 `WRITE`。我们将在下一节详细介绍这两种锁类型。

要释放表的锁，请使用以下语句：

```
UNLOCK TABLES;
```

表锁定为READ

表的`READ`锁具有以下功能：

- 同时可以通过多个会话获取表的 `READ` 锁。此外，其他会话可以从表中读取数据，而无需获取锁定。
- 持有 `READ` 锁的会话只能从表中读取数据，但不能写入。此外，其他会话在释

放 `READ` 锁之前无法将数据写入表中。来自另一个会话的写操作将被放入等待状态，直到释放 `READ` 锁。

- 如果会话正常或异常终止，MySQL 将会隐式释放所有锁。这也与 `WRITE` 锁相关。

下面我们来看看在以下情况下 `READ` 锁如何工作。

首先，连接到 `testdb` 数据库。要查找当前的连接ID，请使用 `CONNECTION_ID()` 函数，如下所示：

```
mysql> SELECT CONNECTION_ID();
+-----+
| CONNECTION_ID() |
+-----+
|          6      |
+-----+
1 row in set
```

然后，在向 `tbl` 表中插入一个新行。

```
INSERT INTO tbl(col) VALUES(10);
```

接下来，从上表 `tbl` 中检索所有行。

```
mysql> SELECT * FROM tbl;
+---+---+
| id | col |
+---+---+
| 1  | 10  |
+---+---+
1 row in set
```

之后，要获取锁，可以使用 `LOCK TABLE` 语句。最后，在同一个会话中，如果您尝试在 `tbl` 表中插入一个新行，将收到一条错误消息。

```
mysql> LOCK TABLE tbl READ;
Query OK, 0 rows affected

mysql> INSERT INTO tbl(col) VALUES(11);
1099 - Table 'tbl' was locked with a READ lock and can't be updated
mysql>
```

所以一旦获得了 `READ` 锁定，就不能在同一个会话中的表中写入数据。让我们从不同的会话中来查看 `READ` 锁。

首先，打开另一个终端并连接到数据库 `testdb`，然后检查连接ID：

```
mysql> SELECT CONNECTION_ID();
+-----+
| CONNECTION_ID() |
+-----+
| 7 |
+-----+
1 row in set
```

然后，从 `tbl` 检索数据，如下所示 -

```
mysql> SELECT * FROM tbl;
+---+---+
| id | col |
+---+---+
| 1 | 10 |
+---+---+
1 row in set
```

接下来，从第二个会话(会话ID为 `7`)插入一个新行到 `tbl` 表中。

 3 12:02:43 insert into tbl(col) values(20) Running...

第二个会话的插入操作处于等待状态，因为第一个会话已经在 `tbl` 表上获取了一个 `READ` 锁，并且尚未释放。

可以使用 `SHOW PROCESSLIST` 语句查看详细信息，如下所示 -

```
mysql> SHOW PROCESSLIST;
+-----+-----+-----+-----+-----+-----+
| Id | User | Host | db | Command | Time | Stat |
+-----+-----+-----+-----+-----+-----+
| 2 | root | localhost:51998 | NULL | Sleep | 474 | NULL |
| 3 | root | localhost:51999 | yiibaidb | Sleep | 3633 | NULL |
| 6 | root | localhost:52232 | testdb | Query | 0 | starting |
| 7 | root | localhost:53642 | testdb | SHOW PROCESSLIST |
| 7 | root | localhost:53642 | testdb | Query | 110 | Waiting for table metadata lock |
| 7 | root | localhost:53642 | testdb | INSERT INTO tbl(col) VALUES(20) |
+-----+-----+-----+-----+-----+-----+
4 rows in set
```

之后，返回第一个会话并使用 `UNLOCK TABLES` 语句来释放锁。从第一个会话释放 `READ` 锁之后，在第二个会话中执行 `INSERT` 操作。

最后，查看 `tbl` 表中的数据，以查看第二个会话中的 `INSERT` 操作是否真的执行。

```
mysql> SELECT * FROM tbl;
+---+---+
| id | col |
+---+---+
| 1 | 10 |
| 2 | 20 |
+---+---+
2 rows in set
```

将MySQL表锁定WRITE

表锁为 `WRITE` 具有以下功能：

- 只有拥有表锁定的会话才能从表读取和写入数据。

- 在释放 WRITE 锁之前，其他会话不能从表中读写。

详细了解 WRITE 锁的工作原理。

首先，从第一个会话获取一个 WRITE 锁。

```
LOCK TABLE tbl WRITE;
```

然后，在 `tbl` 表中插入一个新行。

```
INSERT INTO tbl(col) VALUES(11);
```

没有问题，上面语句可能正常执行。接下来，从 `tbl` 表读取数据。

```
mysql> SELECT * FROM tbl;
+----+---+
| id | col |
+----+---+
| 1  | 10  |
| 2  | 20  |
| 3  | 11  |
+----+---+
3 rows in set
```

之后，打开第二个连接到MySQL的会话，尝试写和读数据：

MySQL将这些操作置于等待状态。可以在第一个会话中，使用 `SHOW PROCESSLIST` 语句来查看它。

```
mysql> SHOW PROCESSLIST;
+-----+-----+-----+-----+-----+-----+
| Id | User | Host           | db   | Command | Time | Sta |
te
+-----+-----+-----+-----+-----+-----+
| 2  | root  | localhost:51998 | NULL | Sleep   | 8477 |     |
| 3  | root  | localhost:51999 | yiibaidb | Sleep  | 11636 |     |
| 8  | root  | localhost:54012 | testdb | Sleep  | 119  |     |
| 9  | root  | localhost:54013 | testdb | Query  | 0    | sta |
ting
| 10 | root  | localhost:54016 | testdb | Query  | 49   | Wai |
ting for table metadata lock | INSERT INTO tbl(col) VALUES(21)
+-----+-----+-----+-----+-----+-----+
5 rows in set
```

最后，从第一个会话释放锁。执行以下语句 -

```
UNLOCK TABLES;
```

执行上面语句后，将看到第二个会话中的所有待处理已经执行操作。

```
SELECT * FROM tbl;
Query OK, 1 row affected

+-----+
| id | col |
+-----+
| 1  | 10  |
| 2  | 20  |
| 3  | 11  |
| 4  | 21  |
+-----+
4 rows in set
```

在本教程中，我们向您展示了如何锁定和解锁：READ 和 WRITE 操作，以便在会话之间配合表访问。

在本教程中，您将学习如何在MySQL中管理数据库。例如，学习如何创建新的数据库，删除现有数据库以及显示MySQL数据库服务器中的所有数据库。

下面让我们演示如何在MySQL中创建一个新的数据库。

创建数据库

在与数据进行任何其他操作之前，需要创建一个数据库。数据库是数据的容器。它可用于存储联系人，供应商，客户或任何想存储的数据。在MySQL中，数据库是用于存储和操作诸如表，数据库视图，触发器，存储过程等数据的对象的集合。

要在MySQL中创建数据库，请使用 `CREATE DATABASE` 语句，如下：

```
CREATE DATABASE [IF NOT EXISTS] database_name;
```

我们来更详细地看看 `CREATE DATABASE` 语句：

- `CREATE DATABASE` 语句的后面是要创建的数据库名称。建议数据库名称尽可能是有意义和具有一定的描述性。
- `IF NOT EXISTS` 是语句的可选子句。`IF NOT EXISTS` 子句可防止创建数据库服务器中已存在的新数据库的错误。不能在MySQL数据库服务器中具有相同名称的数据库。

例如，要创建一个名称为 `mytestdb` 数据库，可以执行 `CREATE DATABASE` 语句后接数据库名称：`mytestdb`，如果当前MySQL服务器中没有数据库：`mytestdb`，则创建成功，如下所示：

```
CREATE DATABASE IF NOT EXISTS mytestdb;
```

执行此语句后，MySQL返回一条消息，通知新数据库是否已成功创建。

显示数据库

`SHOW DATABASES` 语句显示MySQL数据库服务器中的所有数据库。您可以使用 `SHOW DATABASES` 语句来查看您要创建的数据库，或者在创建新数据库之前查看数据库服务器上的所有数据库，例如：

```
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| testdb |  
| yiibaidb |  
+-----+  
5 rows in set
```

在此MySQL数据库服务器中有 6 个数据库。

`information_schema` , `performance_schema` 和 `mysql` 是我们安装MySQL时可用的默认数据库，而 `yiibaidb` 是创建的新数据库。

选择要使用的数据库

在使用指定数据库之前，必须通过使用 `USE` 语句告诉MySQL要使用哪个数据库。

```
USE database_name;
```

您可以使用 `USE` 语句选择示例数据库(`yiibaidb`)，如下所示：

```
USE yiibaidb;
```

从现在开始，所有操作(如查询数据，创建新表或调用存储过程)都将对当前数据库(即 `yiibaidb`)产生影响。

删除数据库

删除数据库意味着数据库中的所有数据和关联对象将被永久删除，并且无法撤消。因此，用额外的注意事项执行此查询是非常重要的。

要删除数据库，请使用 `DROP DATABASE` 语句，如下所示：

```
DROP DATABASE [IF EXISTS] database_name;
```

遵循 `DROP DATABASE` 是要删除的数据库名称。与 `CREATE DATABASE` 语句类似，`IF EXISTS` 是该语句的可选部分，以防止您删除数据库服务器中不存在的数据库。

如果要使用 `DROP DATABASE` 语句练习，可以创建一个新数据库，然后将其删除。来看下面的查询：

```
CREATE DATABASE IF NOT EXISTS tempdb;
SHOW DATABASES;
DROP DATABASE IF EXISTS tempdb;
```

三个语句的说明如下：

- 首先，使用 `CREATE DATABASE` 语句创建了一个名为 `tempdb` 的数据库。
- 第二，使用 `SHOW DATABASES` 语句显示所有数据库。
- 第三，使用 `DROP DATABASE` 语句删除了名为 `tempdb` 的数据库。

在本教程中，您学习了各种语句来管理MySQL中的数据库，包括创建新数据库，删除现有数据库，选择要使用的数据库，以及在MySQL数据库服务器中显示所有数据库。

在本教程中，您将学习各种MySQL表类型或存储引擎。了解MySQL中每个表类型的功能至关重要，以便您可以有效地使用它们来最大限度地提高数据库的性能。

MySQL为其表提供了各种存储引擎，它们分别如下所示：

- MyISAM
- InnoDB
- MERGE
- MEMORY(HEAP)
- ARCHIVE
- CSV
- FEDERATED

每个存储引擎都有自己的优点和缺点。了解每个存储引擎功能至关重要，并为表选择最合适的功能，以最大限度地提高数据库的性能。在以下部分中，我们将讨论每个存储引擎及其功能，以便您可以决定使用哪个引擎合适。

MyISAM

*MyISAM*扩展了以前的*ISAM*存储引擎。*MyISAM*表针对压缩和速度进行了优化。*MyISAM*表也可以在平台和操作系统之间移植。

*MyISAM*表的大小可达 256TB，这个数据里是非常巨大的。此外，*MyISAM*表可以压缩为只读表以节省空间。在启动时，MySQL会检查*MyISAM*表是否有损坏，甚至在出现错误的情况下修复它们。*MyISAM*表不是事务安全的。

在MySQL 5.5之前，*MyISAM*是创建表但是不用明确指定存储引擎时的默认存储引擎。从版本5.5起，MySQL使用*InnoDB*作为默认存储引擎。

InnoDB

*InnoDB*表完全支持符合ACID和事务。它们也是性能最佳的。*InnoDB*表支持外键，提交，回滚，前滚操作。*InnoDB*表的大小最多可达 64TB。

像*MyISAM*一样，*InnoDB*表可以在不同的平台和操作系统之间移植。如果需要，MySQL还会在启动时检查和修复*InnoDB*表。

MERGE

MERGE表是将具有相似结构的多个**MyISAM**表组合到一个表中的虚拟表。**MERGE**存储引擎也被称为**MRG_MyISAM**引擎。**MERGE**表没有自己的索引；它会使用组件表的索引。

使用**MERGE**表，可以在连接多个表时加快性能。MySQL只允许您对**MERGE**表执行**SELECT**，**DELETE**，**UPDATE**和**INSERT**操作。如果在 **MERGE** 表上使用 **DROP TABLE** 语句，则仅删除 **MERGE** 规范。基础表不会受到影响。

Memory

内存表存储在内存中，并使用散列索引，使其比**MyISAM**表格快。内存表数据的生命周期取决于数据库服务器的正常运行时间。内存存储引擎以前称为**HEAP**。

Archive

归档存储引擎允许将大量用于归档目的的记录存储为压缩格式以节省磁盘空间。归档存储引擎在插入时压缩记录，并在读取时使用 **zlib** 库对其进行解压缩。

归档表只允许**INSERT**和 **SELECT** 语句。**ARCHIVE** 表不支持索引，因此需要完整的表扫描来读取行。

CSV

CSV存储引擎以逗号分隔值(CSV)文件格式存储数据。**CSV**表格提供了将数据迁移到非SQL应用程序(如电子表格软件)中的便捷方式。

CSV表不支持 **NULL** 数据类型。此外，读操作需要全表扫描。

FEDERATED

FEDERATED存储引擎允许从远程MySQL服务器管理数据，而无需使用集群或复制技术。本地联合表不存储任何数据。从本地联合表查询数据时，数据将从远程联合表自动拉出。

选择MySQL表类型

11.2 MySQL 表类型

您可以根据各种条件下载以下清单以选择最合适的数据引擎或表类型。请查看以下 PDF文件，了解它们之间的区别：

<http://www.yiibai.com/downloads/mysql/engines-feature.pdf>

在本教程中，您已经学习了MySQL中可用的各种数据引擎或表类型。

在本教程中，我们将向您展示如何使用MySQL `CREATE TABLE` 语句在数据库中创建新表。

MySQL CREATE TABLE语法

要在数据库中创建一个新表，可以使用MySQL `CREATE TABLE` 语句。`CREATE TABLE` 语句是MySQL中最复杂的语句之一。

下面以简单的形式来说明 `CREATE TABLE` 语句的语法：

```
CREATE TABLE [IF NOT EXISTS] table_name(
    column_list
) engine=table_type;
```

我们来更详细地来查看其语法：

- 首先，指定要在 `CREATE TABLE` 子句之后创建的表的名称。表名在数据库中必须是唯一的。`IF NOT EXISTS` 是语句的可选部分，允许您检查正在创建的表是否已存在于数据库中。如果是这种情况，MySQL将忽略整个语句，不会创建任何新的表。强烈建议在每个 `CREATE TABLE` 语句中使用 `IF NOT EXISTS` 来防止创建已存在的新表而产生错误。
- 其次，在 `column_list` 部分指定表的列表。字段的列用逗号(，)分隔。我们将在下一节中向您展示如何更详细地列(字段)定义。
- 第三，需要为 `engine` 子句中的表指定存储引擎。可以使用任何存储引擎，如：`InnoDB`，`MyISAM`，`HEAP`，`EXAMPLE`，`CSV`，`ARCHIVE`，`MERGE`，`FEDERATED`或`NDBCLUSTER`。如果不明确声明存储引擎，MySQL将默认使用`InnoDB`。

注：`InnoDB`自MySQL 5.5之后成为默认存储引擎。`InnoDB`表类型带来了诸如ACID事务，引用完整性和崩溃恢复等关系数据库管理系统的诸多好处。在以前的版本中，MySQL使用`MyISAM`作为默认存储引擎。

要在 `CREATE TABLE` 语句中为表定义列，请使用以下语法：

```
column_name data_type[size] [NOT NULL|NULL] [DEFAULT value]
[AUTO_INCREMENT]
```

以上语法中最重要的组成部分是：

- `column_name` 指定列的名称。每列具有特定数据类型和大小，例如：`VARCHAR(255)`。
- `NOT NULL` 或 `NULL` 表示该列是否接受 `NULL` 值。
- `DEFAULT` 值用于指定列的默认值。
- `AUTO_INCREMENT` 指示每当将新行插入到表中时，列的值会自动增加。每个表都有一个且只有一个 `AUTO_INCREMENT` 列。

如果要将表的特定列设置为主键，则使用以下语法：

```
PRIMARY KEY (col1, col2, ...)
```

MySQL CREATE TABLE语句示例

下面让我们练习一个例子，在示例数据库(testdb)中创建一个名为 `tasks` 的新表，如下所示：

可以使用 `CREATE TABLE` 语句创建这个 `tasks` 表，如下所示：

```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT(11) NOT NULL AUTO_INCREMENT,
    subject VARCHAR(45) DEFAULT NULL,
    start_date DATE DEFAULT NULL,
    end_date DATE DEFAULT NULL,
    description VARCHAR(200) DEFAULT NULL,
    PRIMARY KEY (task_id)
) ENGINE=InnoDB;
```

在本教程中，您已经学习了如何使用MySQL `CREATE TABLE` 语句在数据库中创建新表。

在本教程中，我们将向您展示如何使用MySQL序列为表的ID列自动生成唯一编号。

创建MySQL序列

在MySQL中，序列是以升序生成的整数列表，即 1, 2, 3 ... 许多应用程序需要序列来生成主要用于识别的唯一数字，例如：CRM中的客户ID，HR中的员工编号，服务器管理系统的设备编号等。

要自动在MySQL中创建序列，可以在列上设置 `AUTO_INCREMENT` 属性，这通常是主键列。

使用 `AUTO_INCREMENT` 属性时，将应用以下规则：

- 每个表只有一个 `AUTO_INCREMENT` 列，其数据类型通常为整数。
- 必须对 `AUTO_INCREMENT` 列进行索引，它可以是 `PRIMARY KEY` 或 `UNIQUE` 索引。
- `AUTO_INCREMENT` 列必须具有 `NOT NULL` 约束。当您为列设置 `AUTO_INCREMENT` 属性时，MySQL会自动将 `NOT NULL` 约束隐式添加到列中。

创建MySQL序列示例

以下语句创建一个名为 `employees` 的表，其 `emp_no` 列为 `AUTO_INCREMENT` 列：

```
USE testdb;
CREATE TABLE employees(
    emp_no INT(4) AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name  VARCHAR(50)
);
```

MySQL序列如何工作

`AUTO_INCREMENT` 列具有以下属性：

- `AUTO_INCREMENT` 列的起始值为 1，当您向列中插入 `NULL` 值或在 `INSERT` 语句中省略其值时，它将增加 1。

- 要获取最后生成的序列号，请使用 `LAST_INSERT_ID()` 函数。我们经常要在后续语句中使用最后一个插入 ID，例如将数据插入到表中。最后生成的序列在会话中是唯一的。换句话说，如果另一个连接生成序列号，从连接中可以使用 `LAST_INSERT_ID()` 函数获取它。
- 如果将新行插入到表中并指定序列列的值，如果序列号不存在于列中，则 MySQL 将插入序列号，如果序列号已存在，则会发出错误。如果插入大于下一个序列号的新值，MySQL 将使用新值作为起始序列号，并生成大于当前值的唯一序列号。这会在序列中产生一段空白(不连续)。
- 如果使用 `UPDATE` 语句将 `AUTO_INCREMENT` 列中的值更新为已存在的值，如果该列具有唯一索引，则 MySQL 将发出重复键错误。如果将 `AUTO_INCREMENT` 列更新为大于列中现有值的值，MySQL 将使用最后一个插入序列号加 1 的值作为下一行列号值。例如，如果最后一个插入序列号为 3，然后又将其更新为 10，那么新插入行的序列号不是 11，而是 4。
- 如果使用 `DELETE` 语句删除最后插入的行，MySQL 可能会也可能不会根据表的存储引擎重复使用已删除的序列号。如果您删除一行，则 `MyISAM` 表不会重复使用已删除的序列号，例如，如果删除表中的最后一个插入 ID 为 10，则 MySQL 仍会为新行生成 11 个下一个序列号。与 `MyISAM` 表类似，`InnoDB` 表在行被删除时不重复使用序列号。

在列上设置 `AUTO_INCREMENT` 属性后，可以以各种方式重置自动增量值，例如使用 `ALTER TABLE` 语句。

我们来看一下一些例子来更好地了解 MySQL 序列。

第一步，在 `employees` 表中插入两行：

```
INSERT INTO employees(first_name, last_name)
VALUES('John', 'Doe'),
      ('Mary', 'Jane');
```

第二步，从 `employees` 表中查询选择数据：

```
mysql> SELECT * FROM employees;
+-----+-----+-----+
| emp_no | first_name | last_name |
+-----+-----+-----+
| 1      | John       | Doe      |
| 2      | Mary       | Jane     |
+-----+-----+-----+
2 rows in set
```

第三步，删除 `emp_no` 为 2 的第二个员工信息：

```
mysql> DELETE FROM employees
WHERE emp_no = 2;
Query OK, 1 row affected

mysql> SELECT * FROM employees;
+-----+-----+-----+
| emp_no | first_name | last_name |
+-----+-----+-----+
| 1      | John       | Doe      |
+-----+-----+-----+
1 row in set
```

第四步，插入新员工，并查询最后一位员工信息(`emp_no`)：

```
mysql> INSERT INTO employees(first_name, last_name)
VALUES('Jack', 'Lee');
Query OK, 1 row affected

mysql> SELECT * FROM employees;
+-----+-----+-----+
| emp_no | first_name | last_name |
+-----+-----+-----+
| 1      | John       | Doe      |
| 3      | Jack       | Lee      |
+-----+-----+-----+
2 rows in set
```

因为 `employees` 表的存储引擎是 *InnoDB*，它不会重复使用已删除的序列号。新行使用 `emp_no` 的值是 3。

第五步，将 `emp_no = 3` 已存在新员工更新为 `emp_no = 1`：

```
UPDATE employees
SET first_name = 'Joe',
    emp_no = 1
WHERE emp_no = 3;
```

上面语句执行时，MySQL发出主键重复条目的错误。如何来解决它？

```
UPDATE employees
SET first_name = 'Joe',
    emp_no = 10
WHERE emp_no = 3;
```

执行结果如下 -

```
mysql> UPDATE employees
SET first_name = 'Joe',
    emp_no = 10
WHERE emp_no = 3;
Query OK, 1 row affected
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> SELECT * FROM employees;
+-----+-----+-----+
| emp_no | first_name | last_name |
+-----+-----+-----+
|     1 | John       | Doe      |
|    10 | Joe        | Lee      |
+-----+-----+-----+
2 rows in set
```

上面步骤中，将序列号更新为 10。

第六，插入新员工数据 -

```
mysql> INSERT INTO employees(first_name, last_name)
VALUES('Wang', 'Lee');
Query OK, 1 row affected
```

```
mysql> SELECT * FROM employees;
+-----+-----+-----+
| emp_no | first_name | last_name |
+-----+-----+-----+
|     1  | John       | Doe      |
|     4  | Wang       | Lee      |
|    10  | Joe        | Lee      |
+-----+-----+-----+
```

最后插入的下一个序列号是 4，因此，MySQL 使用数字是 4 作为新行序列值，而不是 11。

在本教程中，您已经学习了如何使用 MySQL 序列为主键列生成唯一的编号，方法是为列设置 AUTO_INCREMENT 属性。

在本教程中，您将了解并使用 MySQL `ALTER TABLE` 语句来更改现有表结构(如添加或删除列，更改列属性等)。

MySQL ALTER TABLE语句简介

可以使用 `ALTER TABLE` 语句来更改现有表的结构。 `ALTER TABLE` 语句可用来添加列，删除列，更改列的数据类型，添加主键，重命名表等等。以下说明了 `ALTER TABLE` 语句语法：

```
ALTER TABLE table_name action1[,action2,...]
```

要更改现有表的结构：

- 首先，在 `ALTER TABLE` 子句之后指定要更改的表名称。
- 其次，列出一组要应用于该表的操作。操作可以是添加新列，添加主键，重命名表等任何操作。`ALTER TABLE` 语句允许在单个 `ALTER TABLE` 语句中应用多个操作，每个操作由逗号(，)分隔。

让我们创建一个用于练习 `ALTER TABLE` 语句的新表。

我们将在[示例数据库\(yiibaidb\)](#)中创建一个名为 `tasks` 的新表。以下是创建 `tasks` 表的脚本。

```
DROP TABLE IF EXISTS tasks;

CREATE TABLE tasks (
    task_id INT NOT NULL,
    subject VARCHAR(45) NULL,
    start_date DATE NULL,
    end_date DATE NULL,
    description VARCHAR(200) NULL,
    PRIMARY KEY (task_id),
    UNIQUE INDEX task_id_unique (task_id ASC)
);
```

使用MySQL ALTER TABLE语句更改列

使用**MySQL ALTER TABLE**语句来设置列的自动递增属性

假设您希望在任务表中插入新行时，`task_id` 列的值会自动增加 1。那么可以使用 `ALTER TABLE` 语句将 `task_id` 列的属性设置为 `AUTO_INCREMENT`，如下所示：

```
ALTER TABLE tasks
CHANGE COLUMN task_id task_id INT(11) NOT NULL AUTO_INCREMENT;
```

可以通过在 `tasks` 表中插入一些行数据来验证更改。

```
INSERT INTO tasks(subject,
                  start_date,
                  end_date,
                  description)
VALUES('Learn MySQL ALTER TABLE',
       Now(),
       Now(),
       'Practicing MySQL ALTER TABLE statement');

INSERT INTO tasks(subject,
                  start_date,
                  end_date,
                  description)
VALUES('Learn MySQL CREATE TABLE',
       Now(),
       Now(),
       'Practicing MySQL CREATE TABLE statement');
```

您可以[查询数据](#)以查看每次插入新行时 `task_id` 列的值是否增加 1：

```
SELECT
  task_id, description
FROM
  tasks;
```

使用**MySQL ALTER TABLE**语句将新的列添加到表中

由于新的业务需求，需要添加一个名为 `complete` 的新列，以便在任务表中存储每个任务的完成百分比。在这种情况下，您可以使用 `ALTER TABLE` 将新列添加到 `tasks` 表中，如下所示：

```
ALTER TABLE tasks
ADD COLUMN complete DECIMAL(2,1) NULL
AFTER description;
```

使用MySQL ALTER TABLE从表中删除列

假设您不想将任务的描述存储在 `tasks` 表中了，并且必须将其删除。以下语句允许您删除 `tasks` 表的 `description` 列：

```
ALTER TABLE tasks
DROP COLUMN description;
```

使用MySQL ALTER TABLE语句重命名表

可以使用 `ALTER TABLE` 语句重命名表。请注意，在重命名表之前，应该认真考虑以了解更改是否影响数据库和应用程序层，不要因为重命名表之后，应用程序因未找到数据库表而出错。

以下语句将 `tasks` 表重命名为 `work_items` 表：

```
ALTER TABLE tasks
RENAME TO work_items;
```

在本教程中，您学习了如何使用MySQL `ALTER TABLE` 语句来更改现有的表结构并重命名表。

在本教程中，您将学习如何使用MySQL `RENAME TABLE` 语句和 `ALTER TABLE` 语句重命名表。

MySQL RENAME TABLE语句简介

由于业务需求变化，我们需要将当前表重新命名为新表，以更好地反映或表示新情况。MySQL提供了一个非常有用的语句来更改一个或多个表的名称。

要更改一个或多个表，我们使用 `RENAME TABLE` 语句如下：

```
RENAME TABLE old_table_name TO new_table_name;
```

旧表(`old_table_name`)必须存在，新表(`new_table_name`)必须不存在。如果新表`new_table_name`存在，则该语句将失败。

除了表之外，我们还可以使用 `RENAME TABLE` 语句来重命名[视图](#)。

在执行 `RENAME TABLE` 语句之前，必须确保没有活动事务或[锁定表](#)。

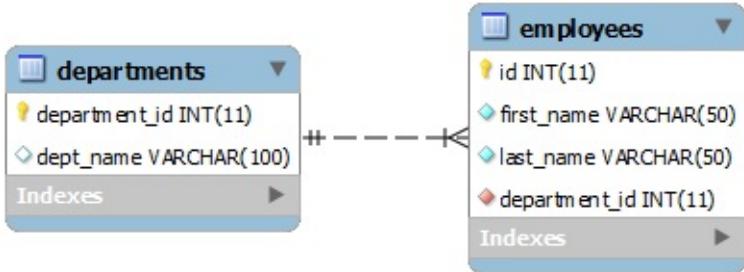
请注意，不能使用 `RENAME TABLE` 语句来重命名[临时表](#)，但可以使用[ALTER TABLE语句](#)重命名临时表。

在安全性方面，我们[授予旧表](#)的任何权限必须手动迁移到新表。

在重命名表之前，应该彻底地评估影响。例如，应该调查哪些应用程序正在使用该表。如果表的名称更改，那么引用表名的应用程序代码也需要更改。此外，您必须手动调整引用该表的其他数据库对象，如[视图](#)，[存储过程](#)，[触发器](#)，[外键约束](#)等。我们将在下面的例子中更详细地讨论。

MySQL RENAME TABLE示例

首先，我们创建一个名为 `hrdb` 的新数据库，它由两个表组成：`employees` 和 `departments`。



创建数据库 -

```
CREATE DATABASE IF NOT EXISTS hrdb;
```

创建表 -

```

USE hrdb;

CREATE TABLE departments (
    department_id INT AUTO_INCREMENT PRIMARY KEY,
    dept_name VARCHAR(100)
);

CREATE TABLE employees (
    id int AUTO_INCREMENT primary key,
    first_name varchar(50) not null,
    last_name varchar(50) not null,
    department_id int not null,
    FOREIGN KEY (department_id)
        REFERENCES departments (department_id)
);
    
```

其次，将样本数据插入到 `employees` 和 `departments` 表中：

```
-- 插入数据到 departments 表中
INSERT INTO departments(dept_name)
VALUES('Sales'),('Marketing'),('Finance'),('Accounting'),('Warehouses'),('Production');

-- 插入数据到 employees 表中
INSERT INTO employees(first_name, last_name, department_id)
VALUES('John', 'Doe', 1),
('Bush', 'Lily', 2),
('David', 'Dave', 3),
('Mary', 'Jane', 4),
('Jonatha', 'Josh', 5),
('Mateo', 'More', 1);
```

第三，查询在 `employees` 和 `departments` 表中的数据：

```

mysql> SELECT
    department_id, dept_name
FROM
    departments;
+-----+-----+
| department_id | dept_name |
+-----+-----+
| 1 | Sales |
| 2 | Marketing |
| 3 | Finance |
| 4 | Accounting |
| 5 | Warehouses |
| 6 | Production |
+-----+-----+
6 rows in set

mysql> SELECT
    id, first_name, last_name, department_id
FROM
    employees;
+-----+-----+-----+-----+
| id | first_name | last_name | department_id |
+-----+-----+-----+-----+
| 1 | John | Doe | 1 |
| 2 | Bush | Lily | 2 |
| 3 | David | Dave | 3 |
| 4 | Mary | Jane | 4 |
| 5 | Jonatha | Josh | 5 |
| 6 | Mateo | More | 1 |
+-----+-----+-----+-----+
6 rows in set

```

重命名视图引用的表

如果重命名一个被视图引用的表，在重命名表后，视图就无效了，并且必须手动调整视图。

例如，我们基于 `employees` 和 `departments` 表创建一个名为 `v_employee_info` 的视图，如下所示：

```
CREATE VIEW v_employee_info AS
SELECT
    id, first_name, last_name, dept_name
FROM
    employees
        INNER JOIN
    departments USING (department_id);
```

视图使用内连接子句来连接 employees 和 departments 表。

以下SELECT语句返回 v_employee_info 视图中的所有数据。

```
mysql> SELECT
    *
    FROM
    v_employee_info;
+----+-----+-----+-----+
| id | first_name | last_name | dept_name |
+----+-----+-----+-----+
| 1  | John       | Doe       | Sales      |
| 2  | Bush       | Lily      | Marketing  |
| 3  | David      | Dave      | Finance    |
| 4  | Mary       | Jane      | Accounting |
| 5  | Jonatha    | Josh      | Warehouses |
| 6  | Mateo      | More      | Sales      |
+----+-----+-----+-----+
6 rows in set
```

现在，将 v_employee_info 视图中的 employees 表重命名为 people，并查询视图的数据。

```
RENAME TABLE employees TO people;

-- 查询数据
SELECT
    *
    FROM
    v_employee_info;
```

MySQL返回以下错误消息：

```
1356 - View 'hrdb.v_employee_info' references invalid table(s) or
       column(s) or function(s) or definer/invoker of view lack rights
       to use them
```

我们可以使用 `CHECK TABLE` 语句来检查 `v_employee_info` 视图的状态如下：

```
CHECK TABLE v_employee_info;
mysql> CHECK TABLE v_employee_info;
+-----+-----+-----+
|-----+-----+-----+
| Table          | Op      | Msg_type | Msg_text
|-----+-----+-----+
|-----+-----+-----+
| hrdb.v_employee_info | check | Error    | Table 'hrdb.employees' doesn't exist
|-----+-----+-----+
| hrdb.v_employee_info | check | Error    | View 'hrdb.v_employee_info' references invalid table(s) or column(s) or function(s)
| or definer/invoker of view lack rights to use them |
| hrdb.v_employee_info | check | error    | Corrupt
|-----+-----+-----+
3 rows in set
```

需要手动更改 `v_employee_info` 视图，以便它引用 `people` 表而不是 `employees` 表。

重命名由存储过程引用的表

如果要重命名由存储过程引用的表，则必须像对视图一样进行手动调整。

首先，将 `people` 表重命名为 `employees` 表。

```
RENAME TABLE people TO employees;
```

然后，创建一个名为 `get_employee` 的新存储过程，该过程引用 `employees` 表。

```
DELIMITER $$

CREATE PROCEDURE get_employee(IN p_id INT)

BEGIN
    SELECT first_name
    , last_name
    , dept_name
    FROM employees
    INNER JOIN departments using (department_id)
    WHERE id = p_id;
END $$

DELIMITER;
```

接下来，执行 `get_employee` 存储过程从 `employees` 表来获取 `id` 为 `1` 的员工的数据，如下所示：

```
CALL get_employee(1);
```

执行上面查询语句，得到以下结果 -

```
mysql> CALL get_employee(1);
+-----+-----+-----+
| first_name | last_name | dept_name |
+-----+-----+-----+
| John      | Doe       | Sales     |
+-----+-----+-----+
1 row in set

Query OK, 0 rows affected
```

之后，我们再次将 `employees` 表重新命名为 `people` 表。

```
RENAME TABLE employees TO people;
```

最后，调用 `get_employee` 存储过程来获取 `id` 为 2 的员工信息：

```
CALL get_employee(2);
```

MySQL 返回以下错误消息：

```
1146 - Table 'hrdb.employees' doesn't exist
```

要解决这个问题，我们必须手动将存储过程中的 `employees` 表更改为 `people` 表。

重命名引用外键的表

`departments` 表使用 `department_id` 列链接到 `employees` 表。
`employees` 表中的 `department_id` 列是引用 `departments` 表的 `department_id` 列作为外键。

如果重命名 `departments` 表，那么指向 `departments` 表的所有外键都不会被自动更新。在这种情况下，我们必须手动删除并重新创建外键。

```
RENAME TABLE departments TO depts;
```

我们删除 ID 为 1 的部门，由于外键约束，`people` 表中的所有行也应删除。但是，我们将 `department` 表重命名为 `depts` 表，而不会手动更新外键，MySQL 会返回错误，如下所示：

```
DELETE FROM depts
WHERE
    department_id = 1;
```

执行上面语句，得到以下错误提示 -

```
1451 - Cannot delete or update a parent row: a foreign key constraint fails (`hrdb`.`people`, CONSTRAINT `people_ibfk_1` FOREIGN KEY (`department_id`) REFERENCES `depts` (`department_id`))
```



重命名多个表

也可以使用 `RENAME TABLE` 语句来一次重命名多个表。见下列声明：

```
RENAME TABLE old_table_name_1 TO new_table_name_2,
          old_table_name_2 TO new_table_name_2, ...
```

以下语句将 `people` 和 `depts` 重命名为 `employees` 和 `departments` 表：

```
RENAME TABLE depts TO departments,
          people TO employees;
```

注意 `RENAME TABLE` 语句不是原子的。所以如果任何时候发生错误，MySQL会将所有重新命名的表都回滚到旧名称。

使用`ALTER TABLE`语句重命名表

我们可以使用 `ALTER TABLE` 语句重命名一个表，如下所示：

```
ALTER TABLE old_table_name
RENAME TO new_table_name;
```

`RENAME TABLE` 语句不能用于重命名临时表，这时就可以使用 `ALTER TABLE` 语句来重命名一个临时表。

重命名临时表示例

首先，我们创建一个临时表，其中包含来自 `employees` 表的 `last_name` 列的所有唯一的姓氏：

```
CREATE TEMPORARY TABLE lastnames
SELECT DISTINCT last_name from employees;
```

第二步，使用 `RENAME TABLE` 重命名姓氏表：

```
RENAME TABLE lastnames TO unique_lastnames;
```

MySQL返回以下错误消息：

```
Error Code: 1017. Can't find file: '.\hrdb\lastnames.frm' (errno : 2 - No such file or directory)
```

第三，使用 `ALTER TABLE` 语句来重命名姓氏表。

```
ALTER TABLE lastnames  
RENAME TO unique_lastnames;
```

第四，从 `unique_lastnames` 临时表查询数据：

```
SELECT  
    last_name  
FROM  
    unique_lastnames;
```

```
+-----+  
| last_name |  
+-----+  
| Doe      |  
| Lily     |  
| Dave     |  
| Jane     |  
| Josh     |  
| More     |  
+-----+  
6 rows in set
```

在本教程中，我们向您展示了如何使用MySQL `RENAME TABLE` 和 `ALTER TABLE` 语句重命名表。

在本教程中，我们将向您展示如何使用MySQL `DROP COLUMN` 语句从表中删除列。

MySQL DROP COLUMN语句简介

在某些情况下，要从表中删除一个或多个列。在这种情况下，可以使用`ALTER TABLE DROP COLUMN`语句如下：

```
ALTER TABLE table  
DROP COLUMN column;
```

下面让我们更详细地学习上面的语法：

- 首先，在`ALTER TABLE`子句之后，指定将要删除的列的表。
- 其次，将列的名称放在`DROP COLUMN`子句之后。

请注意，关键字`COLUMN`是可选的，因此可以使用更短语句书写形式，如下：

```
ALTER TABLE table  
DROP column;
```

要同时从表中删除多个列，请使用以下语法：

```
ALTER TABLE table  
DROP COLUMN column_1,  
DROP COLUMN column_2,  
...;
```

在从表中删除列之前，应该要记住一些重要的事情：

- 从表中删除列会使所有数据库对象(如存储过程，视图，触发器等)依赖于列无效。例如，您可能有一个引用列的存储过程。删除列时，存储过程将变为无效。要修复它，必须手动更改存储过程的代码。
- 取决于已删除列的其他应用程序的代码也必须更改，这需要时间和精力。
- 从大型表中删除列可能会影响数据库的性能。

MySQL DROP COLUMN示例

首先，为了演示创建一个名为 `posts` 的表。

```
USE testdb;

CREATE TABLE posts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    excerpt VARCHAR(400),
    content TEXT,
    created_at DATETIME,
    updated_at DATETIME
);
```

第二，假设要删除 `excerpt` 列，请使用 `ALTER TABLE` 语句如下：

```
ALTER TABLE posts
DROP COLUMN excerpt;
```

第三，要同时删除 `created_at` 和 `updated_at` 列，请使用以下语句：

```
ALTER TABLE posts
DROP COLUMN created_at,
DROP COLUMN updated_at;
```

MySQL删除一列是外键示例

如果您删除作为 [外键](#)的列，MySQL将发出错误。下面我们来一步步演示这个想法。

首先，创建一个名为 `categories` 的表：

```
USE testdb;

CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255)
);
```

第二，将 `categories_id` 列添加到 `posts` 表中。

```
ALTER TABLE posts ADD COLUMN category_id INT NOT NULL;
```

第三，将 `category_id` 列引用 `categories` 表的 `id` 列作为外键。

```
ALTER TABLE posts
ADD CONSTRAINT fk_cat
FOREIGN KEY (category_id)
REFERENCES categories(id) ON DELETE CASCADE;
```

第四，从 `posts` 表中删除 `category_id` 列。

```
ALTER TABLE posts
DROP COLUMN category_id;
```

MySQL发出错误消息：

```
Error Code: 1553. Cannot drop index 'fk_cat': needed in a foreign key constraint
```

为避免此错误，必须删除外键约束才能删除该列。

在本教程中，我们向您展示了如何使用MySQL `DROP COLUMN` 语句从表中删除一个或多个列。

在本教程中，我们将向您展示如何使用 MySQL `ADD COLUMN` 语句将列添加到表中。

MySQL ADD COLUMN语句简介

要向现有表添加新列，请按如下所示使用 `ALTER TABLE ADD COLUMN` 语句：

- 首先，在 `ALTER TABLE` 子句之后指定表名。
- 其次，将新列及其定义放在 `ADD COLUMN` 子句之后。请注意，`COLUMN` 关键字是可选的，因此可以省略它。
- 第三，MySQL 允许通过指定 `FIRST` 关键字将新列添加到表的第一列。它还允许您使用 `AFTER existing_column` 子句在现有列之后添加新列。如果没有明确指定新列的位置，MySQL 会将其添加为最后一列。

要同时向表中添加两个或多个列，请使用以下语法：

```
ALTER TABLE table
ADD [COLUMN] column_name_1 column_1_definition [FIRST|AFTER existing_column],
ADD [COLUMN] column_name_2 column_2_definition [FIRST|AFTER existing_column],
...
;
```

我们来看一些向现有表添加新列的示例。

MySQL ADD COLUMN示例

首先，作为演示目的我们使用以下语句，创建一个名为 `vendor` 的表：

```
USE testdb;

CREATE TABLE IF NOT EXISTS vendors (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255)
);
```

11.8 向表中添加新列

第二，在 `vendors` 表中添加一个名为 `phone` 的新列。因为在 `name` 列之后明确指定了 `phone` 列的位置，MySQL 将会遵守这一点。

```
ALTER TABLE vendors
ADD COLUMN phone VARCHAR(15) AFTER name;
```

第三，在 `vendors` 表中添加一个名为 `vendor_group` 的新列。此时，不指定新列的位置，因此 MySQL 将 `vendor_group` 列添加为 `vendors` 表的最后一列。

```
ALTER TABLE vendors
ADD COLUMN vendor_group INT NOT NULL;
```

让我们插入一些行数据到 `vendors` 表。

```
INSERT INTO vendors(name, phone, vendor_group)
VALUES('IBM', '(408)-298-2987', 1);

INSERT INTO vendors(name, phone, vendor_group)
VALUES('Microsoft', '(408)-298-2988', 1);
```

可以查询 `vendors` 表的数据来查看更改。

```
SELECT
    id, name, phone, vendor_group
FROM
    vendors;

+----+-----+-----+-----+
| id | name      | phone        | vendor_group |
+----+-----+-----+-----+
| 1  | IBM       | (408)-298-2987 | 1            |
| 2  | Microsoft | (408)-298-2988 | 1            |
+----+-----+-----+-----+
2 rows in set
```

第四，同时向 `vendors` 表格添加两列：`email` 和 `hourly_rate`。

```
ALTER TABLE vendors
ADD COLUMN email VARCHAR(100) NOT NULL,
ADD COLUMN hourly_rate decimal(10,2) NOT NULL;
```

请注意，`email` 和 `hourly_rate` 列都分配给 `NOT NULL` 值。但是，`vendors` 表已经有数据。在这种情况下，MySQL 将为这些新列使用默认值。

现在，我们来查看 `vendors` 表中的数据，如下 -

```
SELECT
    id, name, phone, vendor_group, email, hourly_rate
FROM
    vendors;

+----+-----+-----+-----+-----+
| id | name      | phone          | vendor_group | email        | hourly_rate |
+----+-----+-----+-----+-----+
| 1  | IBM       | (408)-298-2987 | 1            |             | 0           |
| 2  | Microsoft | (408)-298-2988 | 1            |             | 0           |
+----+-----+-----+-----+-----+
2 rows in set
```

`email` 列填充有空白字符串值，而不是 `NULL` 值。`hourly_rate` 列填充 `0.00` 值。

如果不小心添加了表中已经存在的列，MySQL 将发出错误。例如，如果执行以下语句：

```
ALTER TABLE vendors
ADD COLUMN vendor_group INT NOT NULL;
```

MySQL发出错误消息：

```
Error Code: 1060. Duplicate column name 'vendor_group'
```

对于具有几列的表，很容易看出哪些列已经存在。但是，有一个大表中有一百个列，这是比较困难的。

在某些情况下，您需要在添加表之前检查一列是否已经存在。但是，没有像 `ADD COLUMN IF NOT EXISTS` 这样的声明。但是可以从 `information_schema` 数据库的 `columns` 表中获取以下信息：

```
SELECT
    IF(count(*) = 1, 'Exist', 'Not Exist') AS result
FROM
    information_schema.columns
WHERE
    table_schema = 'testdb'
    AND table_name = 'vendors'
    AND column_name = 'phone';

+-----+
| result |
+-----+
| Exist  |
+-----+
1 row in set
```

在 `WHERE子句` 中，我们传递了三个参数：表模式或数据库，表名和列名。使用 `IF 函数` 返回列是否存在。

在本教程中，已经学习了如何使用MySQL `DROP COLUMN` 语句将一个或多个列添加到表中。

在本教程中，我们将向您展示如何使用MySQL `DROP TABLE` 语句删除数据中存在的表。

MySQL DROP TABLE语句语法

要删除现有表，请使用MySQL `DROP TABLE` 语句。`DROP TABLE` 的语法如下：

```
DROP [TEMPORARY] TABLE [IF EXISTS] table_name [, table_name] ...
[RESTRICT | CASCADE]
```

`DROP TABLE` 语句从数据库中永久删除一个表及其数据。在MySQL中，也可以使用单个 `DROP TABLE` 语句删除多个表，每个表之间用逗号(，)分隔。

`TEMPORARY` 标志指定仅删除临时表。用于以确保不会意外删除非临时表时非常方便。

`IF EXISTS` 添加可帮助防止删除不存在的表。当使用 `IF EXISTS` 添加时，MySQL生成一个“提示信息”，可以使用 `SHOW WARNING` 语句检索。重要的是要注意，当在指定删除表的列表中存在不存在的表时，`DROP TABLE` 语句将删除所有现有表并发出错误消息或“提示信息”。

如上所述，`DROP TABLE` 语句仅删除表及其数据。但是，它不会删除与表关联的特定用户权限。因此，如果在此之后重新创建了具有相同名称的表，则现有权限将适用于新表，这可能会导致安全风险。

`RESTRICT` 和 `CASCADE` 标志为未来版本的MySQL保留。最后但并非最不重要的 是，必须具有要删除的表的 `DROP` 权限。

MySQL DROP TABLE示例

我们将删除在使用`CREATE TABLE`语句创建表教程中创建的 `tasks` 表。另外，删除一个不存在的表来练习 `SHOW WARNING` 语句。删除 `tasks` 表的语句和删除不存在的表 `nonexistent_table`，如下：

```
DROP TABLE IF EXISTS tasks, nonexistent_table;
```

如果查看数据库，将看到 `tasks` 表已被删除。可以通过使用 `SHOW WARNING` 语句来检查MySQL由于不存在的表生成的“提示信息”，如下所示：

```
SHOW WARNINGS;
```

执行上面语句，得到以下结果 -

```
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message
+-----+-----+-----+
| Note | 1051 | Unknown table 'testdb.nonexistent_table'
+-----+-----+-----+
1 row in set
```

MySQL DROP TABLE WITH LIKE

假设有许多表的名称在数据库中是以字符 `test` 开始，我们希望通过使用单个 `DROP TABLE` 语句删除所有这些表以节省时间。不幸的是，MySQL不提供 `DROP TABLE LIKE` 语句，可以根据以下模式匹配来删除表：

```
DROP TABLE LIKE '%pattern%'
```

但是，有一些解决方法。我们将在此讨论其中一个，以供您参考。为了演示，我们需要创建一些以字符串 `test*` 开头的表。

```
CREATE TABLE IF NOT EXISTS test1(
    id int(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY(id)
);

CREATE TABLE IF NOT EXISTS test2 LIKE test1;
CREATE TABLE IF NOT EXISTS test3 LIKE test1;
CREATE TABLE IF NOT EXISTS test4 LIKE test1;
```

使用相同的表结构创建了名为： test1，test2，test3 和 test4 的四个表。假设您想要一次删除所有字符串 test* 开头的表，可以按照以下步骤：

首先，声明接受数据库模式的两个变量和要与表进行匹配的模式：

```
-- set table schema and pattern matching for tables  
SET @schema = 'testdb';  
SET @pattern = 'test%';
```

接下来，需要构建动态 DROP TABLE 语句：

```
-- build dynamic sql (DROP TABLE tbl1, tbl2...;)  
SELECT CONCAT('DROP TABLE ', GROUP_CONCAT(CONCAT(@schema, '.', table_name)), ';')  
INTO @droplike  
FROM information_schema.tables  
WHERE @schema = database()  
AND table_name LIKE @pattern;
```

基本上，查询指示MySQL转到 information_schema 表，其中包含所有数据库中所有表的数据，并将与模式 @pattern(test%) 匹配的数据
库 @schema(testdb) 中的所有表连接到前缀 DROP TABLE 。GROUP_CONCAT
函数创建一个逗号分隔的表列表。

然后，可以显示动态SQL以验证其是否正常工作：

```
-- display the dynamic sql statement  
SELECT @droplike;
```

执行上面语句，得到以下结果 -

```
mysql> SELECT @droplike;
+-----+
| @droplike |
+-----+
| DROP TABLE testdb.test1,testdb.test2,testdb.test3,testdb.test4; |
+-----+
--+
1 row in set
```

可以看到它按预期工作。

之后，可以使用MySQL中的[prepare语句](#)执行语句如下：

```
-- execute dynamic sql
PREPARE stmt FROM @droplike;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

有关MySQL准备语句的更多信息，请查看[MySQL准备语句教程](#)。

把它们放在一起，如下所示 -

```
-- set table schema and pattern matching for tables
SET @schema = 'testdb';
SET @pattern = 'test%';

-- build dynamic sql (DROP TABLE tbl1, tbl2...; )
SELECT CONCAT('DROP TABLE ', GROUP_CONCAT(CONCAT(@schema, '.', table_name)), ';')
INTO @droplike
FROM information_schema.tables
WHERE @schema = database()
AND table_name LIKE @pattern;

-- display the dynamic sql statement
SELECT @droplike;

-- execute dynamic sql
PREPARE stmt FROM @dropcmd;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

因此，如果要删除在数据库中具有特定模式的多个表，则只需使用上面的脚本来节省时间。所有您需要做的是在 `@pattern` 和 `@schema` 变量中替换模式和数据库模式。如果您经常需要处理此任务，则可以始终基于脚本开发存储过程，并重新使用此存储过程。

在本教程中，我们向您展示了如何使用 `DROP TABLE` 语句来删除特定数据库中的现有表。还讨论了一种解决方法，允许使用 `DROP TABLE` 语句根据模式匹配删除表。

在本教程中，我们将讨论MySQL临时表，并演示如何创建，使用和删除临时表。

MySQL临时表简介

在MySQL中，临时表是一种特殊类型的表，它允许您存储一个临时结果集，可以在单个会话中多次重用。

当使用`JOIN`子句查询需要单个`SELECT`语句的数据是不可能或遇到瓶颈的时候，临时表非常方便。在这种情况下，我们就可以使用临时表来存储直接结果，并使用另一个查询来处理它。

MySQL临时表具有以下特殊功能：

- 使用`CREATE TEMPORARY TABLE`语句创建临时表。请注意，在`CREATE`和`TABLE`关键字之间添加`TEMPORARY`关键字。
- 当会话结束或连接终止时，MySQL会自动删除临时表。当您不再使用临时表时，也可以使用`DROP TABLE`语句来显式删除临时表。
- 一个临时表只能由创建它的客户机访问。不同的客户端可以创建具有相同名称的临时表，而不会导致错误，因为只有创建临时表的客户端才能看到它。但是，在同一个会话中，两个临时表不能共享相同的名称。
- 临时表可以与数据库中的普通表具有相同的名称。例如，如果在[示例数据库\(yiibaidb\)](#)中创建一个名为`employees`的临时表，则现有的`employees`表将变得无法访问。对`employees`表发出的每个查询现在都是指`employees`临时表。当删除您临时表时，永久`employees`表可以再次访问。

注意：即使临时表可以与永久表具有相同的名称，但不推荐。因为这可能会导致混乱，并可能导致意外的数据丢失。例如，如果与数据库服务器的连接丢失，并且您自动重新连接到服务器，则不能区分临时表和永久性表。然后，你又发出一个`DROP TABLE`语句，这个时候删除的可能是永久表而不是临时表，这种结果是不可预料的。

创建MySQL临时表

要创建临时表，只需要将`TEMPORARY`关键字添加到`CREATE TABLE`语句的中间。

例如，以下语句创建一个临时表，按照收入存储前 10 名客户：

```

CREATE TEMPORARY TABLE top10customers
SELECT p.customerNumber,
       c.customerName,
       FORMAT(SUM(p.amount),2) total
  FROM payments p
 INNER JOIN customers c ON c.customerNumber = p.customerNumber
 GROUP BY p.customerNumber
 ORDER BY total DESC
 LIMIT 10;

```

现在，可以从 `top10customers` 临时表中查询数据，例如：

```
SELECT * FROM top10customers;
```

删除MySQL临时表

您可以使用 `DROP TABLE` 语句来删除临时表，但最好添加 `TEMPORARY` 关键字如下：

```
DROP TEMPORARY TABLE table_name;
```

`DROP TEMPORARY TABLE` 语句仅删除临时表，而不是永久表。当将临时表命名为永久表的名称相同时，它可以避免删除永久表的错误。

例如，要删除 `top10customers` 临时表，请使用以下语句：

```
DROP TEMPORARY TABLE top10customers;
```

请注意，如果尝试使用 `DROP TEMPORARY TABLE` 语句删除永久表，则会收到一条错误消息，指出您尝试删除的表是未知的。

如果开发使用连接池或持久连接的应用程序，则不能保证临时表在应用程序终止时自动删除。

因为应用程序使用的数据库连接可能仍然打开并放置在其他客户端使用的连接池中。因此，当不再使用它们时马上删除临时表，这是一个很好的做法。

在本教程中，您已经了解了MySQL临时表以及如何管理临时表，例如创建和删除新临时表。

在本教程中，您将学习如何使用MySQL `TRUNCATE TABLE` 语句删除表中的所有数据。

MySQL TRUNCATE TABLE语句简介

MySQL `TRUNCATE TABLE` 语句允许您删除表中的所有数据。因此，在功能方面，`TRUNCATE TABLE` 语句就像没有`WHERE子句`的`DELETE语句`。但是，在某些情况下，MySQL `TRUNCATE TABLE` 语句比 `DELETE` 语句更有效。

MySQL `TRUNCATE TABLE` 语句的语法如下：

```
TRUNCATE TABLE table_name;
```

在 `TRUNCATE TABLE` 子句后面指定要删除所有数据的表名称。

`TABLE` 关键字是可选的。但是，应该使用它来区分 `TRUNCATE TABLE` 语句和 `TRUNCATE` 函数。

如果您使用[InnoDB表](#)，MySQL将在删除数据之前检查表中是否有可用的外键约束。以下是一些情况：

- 如果表具有任何外键约束，则 `TRUNCATE TABLE` 语句会逐个删除行。如果外键约束具有 `DELETE CASCADE` 动作，则子表中的相应行也将被删除。
- 如果外键约束没有指定`DELETE CASCADE`动作，则 `TRUNCATE TABLE` 将逐个删除行，并且遇到由子表中的行引用的行时，它将停止并发出错误。
- 如果表没有任何外键约束，则 `TRUNCATE TABLE` 语句将删除该表并重新创建一个具有相同结构的新表，这比使用 `DELETE` 语句特别是对于大表更快更有效。

如果您使用其他[存储引擎](#)，则 `TRUNCATE TABLE` 语句将删除并重新创建一个新表。

请注意，如果表具有 `AUTO_INCREMENT` 列，则 `TRUNCATE TABLE` 语句将自动递增值重置为零。

此外，`TRUNCATE TABLE` 语句不使用 `DELETE` 语句，因此与表关联的`DELETE触发器`将不被调用。

MySQL TRUNCATE TABLE示例

首先，创建一个名为 books 的新表：

```
CREATE DATABASE IF NOT EXISTS testdb;

USE testdb;

CREATE TABLE books(
    id int auto_increment primary key,
    title varchar(255) not null
)ENGINE=InnoDB;
```

接下来，使用以下存储过程填充 books 表的数据：

```
DELIMITER $$

CREATE PROCEDURE load_book_data(IN num int(10))
BEGIN
    DECLARE counter int(10) default 0;
    DECLARE book_title varchar(255) default '';

    WHILE counter < num DO
        SET book_title = concat('Book title #',counter);
        SET counter = counter + 1;

        INSERT INTO books(title) VALUES(book_title);
    END WHILE;
END$$

DELIMITER ;
-- DROP PROCEDURE load_book_data;
```

然后，将 10,000 行数据插入到 books 表，执行上面语句将需要一段时间。

```
CALL load_book_data(10000);
```

执行上面语句之后，查看 books 表中的数据：

11.11 TRUNCATE TABLE

```
SELECT * FROM books;
```

最后，使用 `TRUNCATE TABLE` 语句来与 `DELETE` 语句相比执行的速度。

```
select now() as start_time;
TRUNCATE TABLE books;
select now() as end_time;
```

上面语句执行结果，如下所示 -

```
mysql> select now() as start_time;
TRUNCATE TABLE books;
select now() as end_time;
+-----+
| start_time |
+-----+
| 2017-07-24 21:31:07 |
+-----+
1 row in set

Query OK, 0 rows affected
```

```
+-----+
| end_time |
+-----+
| 2017-07-24 21:31:08 |
+-----+
1 row in set
```

可以看到整个过程也就差不多 1 秒钟完成，下面再次调用 `CALL load_book_data(10000);`，完成插入数据后使用 `DELETE` 语句，查看执行的时间 -

```
select now() as start_time;
DELETE FROM books;
select now() as end_time;
```

11.11 TRUNCATE TABLE

执行上面语句，得到以下结果 -

```
mysql> select now() as start_time;
DELETE FROM books;
select now() as end_time;
+-----+
| start_time |
+-----+
| 2017-07-24 21:39:42 |
+-----+
1 row in set
```

Query OK, 10000 rows affected

```
+-----+
| end_time |
+-----+
| 2017-07-24 21:39:44 |
+-----+
1 row in set
```

在本教程中，我们向您展示了如何使用MySQL TRUNCATE TABLE 语句从表中有效地删除所有数据，特别是对于拥有大量数据的表。

在本教程中，您将学习如何使用MySQL索引以及如何利用索引来加快数据检索。我们将介绍一些常用和方便的语句，让您可以有效地管理MySQL索引。

数据库索引或索引，有助于加速从表中检索数据。当从表[查询数据](#)时，首先MySQL会检查索引是否存在，然后MySQL使用索引来选择表的精确物理对应行，而不是扫描整个表。

数据库索引类似于书的目录。如果要查找某个主题，首先查找索引，然后打开具有该主题的页面(页码)，而不扫描整本书。

强烈建议您在经常查询数据的表的列上创建索引。请注意，所有[主键](#)列都是表的主索引。

索引有助于加快查询数据，为什么我们不使用所有列的索引？如果为每列创建索引，MySQL必须构建和维护索引表。每当对表的行进行更改时，MySQL必须重建索引，这需要时间以及降低数据库服务器的性能。换句话说：使用索引可以加快查询数据，但是减慢了修改数据的速度。

创建MySQL索引

创建表时经常创建索引。MySQL自动将声明为 `PRIMARY KEY`，`KEY`，`UNIQUE` 或 `INDEX` 的任何列添加到索引。另外，您可以向已经有数据的表添加索引。

要创建索引，可以使用 `CREATE INDEX` 语句。下面说明了 `CREATE INDEX` 语句的语法：

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name  
USING [BTREE | HASH | RTREE]  
ON table_name (column_name [(length)] [ASC | DESC], ...)
```

首先，根据表类型或存储引擎指定索引：

- 对于 `UNIQUE` 索引，MySQL创建一个约束，索引中的所有值都必须是唯一的。除了BDB之外，所有存储引擎都允许重复的 `NULL` 值。
- `FULLTEXT` 索引仅由[MyISAM](#)存储引擎支持，仅在数据类型为 `CHAR`，`VARCHAR` 或 `TEXT` 的列中接受。
- `SPATIAL` 索引支持空间列，可用于[MyISAM](#)存储引擎。另外，列值不能

为 `NULL`。

然后，在 `USING` 关键字之后命名索引及其类型。索引的名称可以是 `BTREE` `HASH` 或 `RTREE`，但必须根据表的存储引擎遵循允许的索引。

以下是具有相应允许的索引类型的表的存储引擎：

存储引擎	允许的索引类型
MyISAM	<code>BTREE</code> , <code>RTREE</code>
InnoDB	<code>BTREE</code>
MEMORY / HEAP	<code>HASH</code> , <code>BTREE</code>
NDB	<code>HASH</code>

第三，声明要添加到索引的表名称和列表列。

在MySQL中创建索引的示例

在[示例数据库\(yiibaidb\)](#)中，可以使用 `CREATE INDEX` 语句将 `employees` 表的 `officeCode` 列添加索引，如下所示：

```
CREATE INDEX index_officeCode ON employees(officeCode)
```

删除索引

除了创建索引之外，还可以使用 `DROP INDEX` 语句来删除索引。有趣的是，`DROP INDEX` 语句也映射到[ALTER TABLE](#)语句。以下是删除索引的语法：

```
DROP INDEX index_name ON table_name
```

例如，如果要删除 `employees` 表的索引 `index_officeCode`，可以执行以下查询：

```
DROP INDEX index_officeCode ON employees
```

在本教程中，您已经了解了索引以及如何管理MySQL索引，包括创建和删除索引。

在本教程中，您将学习如何使用MySQL `UNIQUE` 索引来防止表中一个或多个列中拥有重复的值。

MySQL UNIQUE索引简介

要强制执行一个或多个列的唯一性值，我们经常使用`PRIMARY KEY`约束。

但是，每个表只有一个主键。如果要使用多个列或一组具有唯一值的列，则不能使用主键约束。

幸运的是，MySQL提供了另一种称为 `UNIQUE` 索引的索引，它允许您在一个或多个列中强制实现值的唯一性。与 `PRIMARY KEY` 索引不同，每个表可以有多个 `UNIQUE` 紴引。

要创建一个 `UNIQUE` 索引，请使用`CREATE UNIQUE INDEX`语句如下：

```
CREATE UNIQUE INDEX index_name
ON table_name(index_column_1, index_column_2, ...);
```

在一个或多个列中强制实现值的唯一性的另一种方法是使用唯一约束。创建唯一约束时，MySQL会在幕后创建一个 `UNIQUE` 的索引。

以下语句说明了如何在[创建表](#)时创建唯一约束。

```
CREATE TABLE table_name(
  ...
  UNIQUE KEY(index_column_, index_column_2, ...)
);
```

也可以使用 `UNIQUE INDEX` 而不是 `UNIQUE KEY`。它们被称为相同的东西。

如果要向现有表添加唯一约束，可以使用`ALTER TABLE`语句，如下所示：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name UNIQUE KEY(column_1, column_2, ...);
```

MySQL UNIQUE 索引和NULL

与其他数据库系统不同，MySQL将NULL值视为不同的值。因此，您可以在 UNIQUE 索引中具有多个 NULL 值。

这就是MySQL的设计方式。这不是一个错误，即使它被报告为一个错误。

另一重要的一点是 UNIQUE 约束不适用于除了BDB存储引擎之外的 NULL 值。

MySQL UNIQUE 索引示例

假设想在应用程序中管理联系人。还希望 contacts 表中的 email 列必须是唯一的。

要执行此规则，请在CREATE TABLE语句中创建唯一的约束，如下所示：

```
USE testdb;

CREATE TABLE IF NOT EXISTS contacts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    phone VARCHAR(15) NOT NULL,
    email VARCHAR(100) NOT NULL,
    UNIQUE KEY unique_email (email)
);
```

如果使用 SHOW INDEXES 语句，将看到MySQL为 email 列创建了一个 UNIQUE 索引。

```
SHOW INDEXES FROM contacts;
```

Table	Non_unique	Key_name	Seq_in_index	Column_n	ame	Collation	Cardinality	Sub_part	Packed	Null	Index	_type	Comment	Index_comment
contacts	0	PRIMARY	1	id	A	A	0	NULL	NULL	NULL	BTREE			
contacts	0	unique_email	1	email	A	A	0	NULL	NULL	NULL	BTREE			

2 rows in set

接下来，向 `contacts` 表中插入一行。

```
INSERT INTO contacts(first_name, last_name, phone, email)
VALUES('Max', 'Su', '(+86)-999-9988', 'max.su@yiibai.com');
```

现在，如果您尝试插入电子邮件是 `max.su@yiibai.com` 的行数据，您将收到一条错误消息。

```
INSERT INTO contacts(first_name, last_name, phone, email)
VALUES('Max', 'Su', '(+86)-999-9988', 'max.su@yiibai.com');
```

执行上面语句后，应该会看到以下结果 -

```
1062 - Duplicate entry 'max.su@yiibai.com' for key 'unique_email'
```

假设您不仅希望电子邮件是唯一的，而且 `first_name`，`last_name` 和 `phone` 的组合也是唯一的。在这种情况下，可以使用 `CREATE INDEX` 语句为这些列创建一个 `UNIQUE` 索引，如下所示：

```
CREATE UNIQUE INDEX idx_name_phone  
ON contacts(first_name, last_name, phone);
```

将以下行添加到 `contacts` 表中会导致错误，因为 `first_name`，`last_name` 和 `phone` 的组合已存在。

```
INSERT INTO contacts(first_name, last_name, phone, email)  
VALUES('Max', 'Su', '(+86)-999-9988', 'john.d@yiibai.com');
```

执行上面语句后，应该会看到以下结果 -

```
1062 - Duplicate entry 'Max-Su-(+86)-999-9988' for key 'idx_name  
_phone'
```

可以看到，不可以将重复电话号码插入到表中。

在本教程中，您已经学习了如何使用MySQL `UNIQUE` 索引来防止数据库中的重复值。

在本教程中，您将了解MySQL数据类型以及如何在MySQL中设计数据库时有效地使用它们。

数据库表包含具有特定数据类型(如数字或字符串)的多个列。MySQL提供更多的数据类型，而不仅仅是数字或字符串。MySQL中的每种数据类型都可以通过以下特征来确定：

- 它用来表示数据值。
- 占用的空间以及值是固定长度还是可变长度。
- 数据类型的值可以被索引。
- MySQL如何比较特定数据类型的值。

数据类型	指定值和范围
char	String(0~255)
varchar	String(0~255)
tinytext	String(0~255)
text	String(0~65536)
blob	String(0~65536)
mediumtext	String(0~16777215)
mediumblob	String(0~16777215)
longblob	String(0~4294967295)
longtext	String(0~4294967295)
tinyint	Integer(-128~127)
smallint	Integer(-32768~32767)
mediumint	Integer(-8388608~8388607)
int	Integer(-214847668~214847667)
bigint	Integer(-9223372036854775808~9223372036854775807)
float	decimal(精确到23位小数)
double	decimal(24~54位小数)
decimal	将 double 转储为字符串形式
date	YYYY-MM-DD
datetime	YYYY-MM-DD HH:MM:SS
timestamp	YYYYMMDDHHMMSS
time	HH:MM:SS
enum	选项值之一
set	选项值子集
boolean	tinyint(1)

MySQL 数值数据类型

在MySQL中，您可以找到所有SQL标准数字类型，包括精确数字数据类型和近似数字数据类型，包括整数，定点和浮点数。此外，MySQL还具有用于存储位值的BIT数据类型。数字类型可以是有符号或无符号，但 BIT 类型除外。

下表显示了MySQL中数字类型的总结：

数字类型	描述
TINYINT	一个很小的整数
SMALLINT	一个小的整数
MEDIUMINT	一个中等大小的整数
INT	一个标准整数
BIGINT	一个大整数
DECIMAL	定点数
FLOAT	单精度浮点数
DOUBLE	双精度浮点数
BIT	一个字节字段

MySQL布尔数据类型

MySQL没有内置的 BOOLEAN 或 BOOL 数据类型。所以要表示布尔值，MySQL使用最小的整数类型，也就是 TINYINT(1)。换句话
说，BOOLEAN 和 BOOL 是 TINYINT(1) 的同义词。

MySQL字符串数据类型

在MySQL中，字符串可以容纳从纯文本到二进制数据(如图像或文件)的任何内容。可以通过使用LIKE运算符，正则表达式和全文搜索，根据模式匹配来比较和搜索字符串。

下表显示了MySQL中的字符串数据类型：

字符串类型	描述
char	固定长度的非二进制(字符)字符串
varchar	可变长度的非二进制字符串
BINARY	一个固定长度的二进制字符串
VARBINARY	一个可变长度的二进制字符串
TINYBLOB	一个非常小的BLOB(二进制大对象)
BLOB	一个小的BLOB(二进制大对象)
MEDIUMBLOB	一个中等大小的BLOB(二进制大对象)
LONGBLOB	一个大的BLOB(二进制大对象)
TINYTEXT	一个非常小的非二进制字符串
TEXT	一个小的非二进制字符串
MEDIUMTEXT	一个中等大小的非二进制字符串
LONGTEXT	一个很大的非二进制字符串
ENUM	枚举; 每个列值可以被分配一个枚举成员
SET	集合; 每个列值可以分配零个或多个 SET 成员

MySQL 日期和时间数据类型

MySQL提供日期和时间的类型以及日期和时间的组合。此外，MySQL还支持时间戳数据类型，用于跟踪表的一行中的更改。如果只想存储没有日期和月份的年份数据，则可以使用 YEAR 数据类型。

下表说明了MySQL日期和时间数据类型：

字符串类型	描述
DATE	YYYY-MM-DD 格式的日期值
TIME	hh:mm:ss 格式的时间值
DATETIME	YYYY-MM-DD hh:mm:ss 格式的日期和时间值
TIMESTAMP	YYYY-MM-DD hh:mm:ss 格式的时间戳记值
YEAR	YYYY 或 YY 格式的年值

MySQL 空间数据类型

MySQL 支持许多包含各种几何和地理值的空间数据类型，如下表所示：

字符串类型	描述
GEOMETRY	任何类型的空间值
POINT	一个点(一对X-Y坐标)
LINESTRING	曲线(一个或多个 POINT 值)
POLYGON	多边形
GEOMETRYCOLLECTION	GEOMETRY 值的集合
MULTILINESTRING	LINESTRING 值的集合
MULTIPOINT	POINT 值的集合
MULTIPOLYGON	POLYGON 值的集合

JSON 数据类型

MySQL 5.7.8 版本支持原生 JSON 数据类型，可以更有效地存储和管理 JSON 文档。本机 JSON 数据类型提供 JSON 文档的自动验证和最佳存储格式。

在本教程中，您学习了各种 MySQL 数据类型，可帮助您确定在创建表时应使用哪些数据类型。

在本教程中，您将了解MySQL INT 或整数数据类型以及如何在数据库表设计中使用它。另外，我们将向您展示如何使用表中整数列的显示宽度和 ZEROFILL 属性。

MySQL INT类型简介

在MySQL中，INT 代表整数。整数可以写成没有分数的数值，例如，它可以是 1 , 100 , 4 , -10 等，它不能是 1.2 , 5/3 等。整数可以是零，正和负。

MySQL支持所有标准SQL整数类型 INTEGER 、 INT 和 SMALLINT 。此外，MySQL提供 TINYINT ， MEDIUMINT ， BIGINT 作为标准SQL的扩展。

MySQL INT 数据类型可以是有符号或无符号。下表说明了每个整数类型的特性，包括以字节为单位的存储，最小值和最大值。

Type	Bytes	Signed		Unsigned	
		Min	Max	Min	Max
TINYINT	1	-2 ⁴	2 ⁴	0	2 ⁸
SMALLINT	2	-2 ⁸	2 ⁸	0	2 ¹⁶
MEDIUMINT	3	-2 ¹²	2 ¹²	0	2 ²⁴
INT	4	-2 ¹⁶	2 ¹⁶	0	2 ³²
BIGINT	8	-2 ³²	2 ³²	0	2 ⁶⁴



在列中使用INT

由于整数类型表示精确数字，因此通常将其用作表的主键。此外，INT 列可以具有 AUTO_INCREMENT 属性。

当在 INT AUTO_INCREMENT 列中插入 NULL 值或 0 时，列的值将设置为下一个序列值。请注意，序列值以 1 开始。

在 AUTO_INCREMENT 列中插入不为 NULL 或零的值时，列接受该值。此外，序列将重置为插入值的下一个值。

我们来看一个使用带有 AUTO_INCREMENT 属性的整数列的表的例子。

首先，使用以下语句创建一个名为 `items` 的新表，其中整数列为主键：

```
USE testdb;

CREATE TABLE items (
    item_id INT AUTO_INCREMENT PRIMARY KEY,
    item_text VARCHAR(255)
);
```

可以在上面的CREATE TABLE语句中使用 `INT` 或 `INTEGER`，因为它们是可互换的。每当在 `items` 表中插入一个新行时，`item_id` 列的值将增加 `1`。

接下来，以下INSERT语句在 `items` 表中插入三行数据。

```
INSERT INTO items(item_text)
VALUES('laptop'), ('mouse'), ('headphone');
```

然后，使用以下SELECT语句从 `items` 表查询数据：

```
SELECT
*
FROM
items;
```

之后，插入一个新的行，该行的 `item_id` 列的值要明确指定。

```
INSERT INTO items(item_id, item_text)
VALUES(10, 'Server');
```

因为 `item_id` 列的当前值为 `10`，所以序列将重置为 `11`。如果插入新行，则 `AUTO_INCREMENT` 列将使用 `11` 作为下一个值。

```
INSERT INTO items(item_text)
VALUES('Router');
```

最后，再次查询 `items` 表中的数据，如下结果。

```
SELECT
*
FROM
items;
```

请注意，由于自 MySQL 5.1 版本起，`AUTO_INCREMENT` 列只接受正值。`AUTO_INCREMENT` 列不支持负数值。

MySQL INT 和显示宽度属性

MySQL 提供了一个扩展，允许您指定显示宽度以及 `INT` 数据类型。显示宽度包含在 `INT` 关键字后面的括号内，例如，`INT(5)` 指定一个显示宽度为五位数的 `INT`。

要注意的是，显示宽度属性不能控制列可以存储的值范围。显示宽度属性通常由应用程序用于格式化整数值。MySQL 将显示宽度属性作为返回结果集的元数据。

具有 `ZEROFILL` 属性的 MySQL INT

除了显示宽度之外，MySQL 还提供了非标准的 `ZEROFILL` 属性。在这种情况下，MySQL 将空格替换为零。请参考以下示例。

首先，使用以下语句创建一个名为 `zerofill_tests` 的表：

```
USE testdb;

CREATE TABLE zerofill_tests(
    id INT AUTO_INCREMENT PRIMARY KEY,
    v1 INT(2) ZEROFILL,
    v2 INT(3) ZEROFILL,
    v3 INT(5) ZEROFILL
);
```

其次，在 `zerofill_tests` 表中插入一个新行。

```
INSERT INTO zerofill_tests(v1,v2,v3)
VALUES(1,6,9);
```

第三，从 `zerofill_tests` 表查询数据。

```
SELECT
    v1, v2, v3
FROM
    zerofill_tests;
```

`v1` 列的显示宽度为 `2`，包括 `ZEROFILL`，它的值为 `1`，因此在输出中看到 `01`。MySQL 将第一个空格替换为 `0`。

`v2` 列具有包含 `ZEROFILL` 的显示宽度 `3`。它的值为 `6`，因此将看到有 `00` 作为前导零。

`v3` 列具有包含 `ZEROFILL` 的显示宽度 `5`，它的值为 `9`，因此 MySQL 在输出数字的开头填零为 `0000`。

请注意，如果对整数列使用 `ZEROFILL` 属性，MySQL 将自动将一个 `UNSIGNED` 属性添加到该列。

在本教程中，我们向您展示了如何在表中使用 MySQL `INT` 数据类型，并向您介绍了整数列的显示宽度和 `ZEROFILL` 属性。

在本教程中，我们将向您介绍MySQL `DECIMAL` 数据类型以及如何在数据库表中有效地使用它。

MySQL DECIMAL数据类型简介

MySQL `DECIMAL` 数据类型用于在数据库中存储精确的数值。我们经常将 `DECIMAL` 数据类型用于保留准确精确度的列，例如会计系统中的货币数据。

要定义数据类型为 `DECIMAL` 的列，请使用以下语法：

```
column_name  DECIMAL(P,D);
```

在上面的语法中：

- `P` 是表示有效数字数的精度。`P` 范围为 `1~65`。
- `D` 是表示小数点后的位数。`D` 的范围是 `0 ~ 30`。MySQL要求 `D` 小于或等于(`<=`) `P`。

`DECIMAL(P,D)` 表示列可以存储 `D` 位小数的 `P` 位数。十进制列的实际范围取决于精度和刻度。

与[INT数据类型](#)一样，`DECIMAL` 类型也具有 `UNSIGNED` 和 `ZEROFILL` 属性。如果使用 `UNSIGNED` 属性，则 `DECIMAL UNSIGNED` 的列将不接受负值。

如果使用 `ZEROFILL`，MySQL将把显示值填充到 `0` 以显示由列定义指定的宽度。另外，如果我们对 `DECIMAL` 列使用 `ZERO FILL`，MySQL将自动将 `UNSIGNED` 属性添加到列。

以下示例使用 `DECIMAL` 数据类型定义的一个叫作 `amount` 的列。

```
amount  DECIMAL(6,2);
```

在此示例中，`amount` 列最多可以存储 6 位数字，小数位数为 2 位；因此，`amount` 列的范围是从 `-9999.99` 到 `9999.99`。

MySQL允许使用以下语法：

```
column_name  DECIMAL(P);
```

这相当于：

```
column_name DECIMAL(P,0);
```

在这种情况下，列不包含小数部分或小数点。

此外，我们甚至可以使用以下语法。

```
column_name DECIMAL;
```

在这种情况下，P 的默认值为 10。

MySQL DECIMAL 存储

MySQL 分别为整数和小数部分分配存储空间。MySQL 使用二进制格式存储 DECIMAL 值。它将 9 位数字包装成 4 个字节。

对于每个部分，需要 4 个字节来存储 9 位数的每个倍数。剩余数字所需的存储如下表所示：

剩余数字	位
0	0
1-2	1
3-4	2
5-6	3
7-9	4

例如，DECIMAL(19,9) 对于小数部分具有 9 位数字，对于整数部分具有 19 位 = 10 位数字，小数部分需要 4 个字节。整数部分对于前 9 位数字需要 4 个字节，1 个剩余字节需要 1 个字节。DECIMAL(19,9) 列总共需要 9 个字节。

MySQL DECIMAL 数据类型和货币数据

经常使用 DECIMAL 数据类型的货币数据，如价格，工资，账户余额等。如果要设计一个处理货币数据的数据库，则可参考以下语法 -

```
amount DECIMAL(19, 2);
```

但是，如果您要遵守公认会计原则(GAAP)规则，则货币栏必须至少包含 4 位小数，以确保舍入值不超过 \$0.01。在这种情况下，应该定义具有 4 位小数的列，如下所示：

```
amount DECIMAL(19, 4);
```

MySQL DECIMAL 数据类型示例

首先，创建一个名为 material 的新表，其中包含三列： id ， description 和 cost 。

```
CREATE TABLE materials (
    id INT AUTO_INCREMENT PRIMARY KEY,
    description VARCHAR(255),
    cost DECIMAL(19 , 4 ) NOT NULL
);
```

第二步，将资料插入 materials 表。

```
INSERT INTO materials(description,cost)
VALUES('Bicycle', 500.34), ('Seat', 10.23), ('Break', 5.21);
```

第三步，从 materials 表查询数据。

```
SELECT
    *
FROM
    materials;
```

第四步，更改 cost 列以包含 ZEROFILL 属性。

```
ALTER TABLE materials
MODIFY cost DECIMAL(19,4) zerofill;
```

第五步，再次查询 `materials` 表。

```
SELECT
*
FROM
materials;
```

如上所见，在输出值中填充了许多零。

在本教程中，我们向您提供了有关MySQL `DECIMAL` 数据类型的详细信息，并向您展示了如何将其应用于存储精确数字数据(例如财务数据)的列。

本教程将向您介绍如何存储和使用位值的MySQL BIT 数据类型。

MySQL BIT数据类型简介

MySQL提供了允许您存储位值的 BIT 类型。 BIT(m) 可以存储多达 m 位的值， m 的范围在 1 到 64 之间。

如果省略，默认值为 1 。所以下列陈述是一样的：

```
column_name BIT(1);
```

以及，

```
column_name BIT;
```

要指定一个位值字面值，可使用 b'val' 或 0bval 来表示，该 val 是仅包含 0 和 1 的二进制值。

开头字符 b 可以写成 B ，例如，以下两种方式都一样：

```
b01  
B11
```

上面书写方式都是有效的位字面量。

但是，前导 0b 是区分大小写的，所以不能使用 0B 。以下是无效的字面值：

```
0B'1000'
```

默认情况下，位值文字的字符集是二进制字符串，如下所示：

```
SELECT CHARSET(B''); -- binary
```

执行结果如下 -

```
mysql> SELECT CHARSET(B'');
+-----+
| CHARSET(B'') |
+-----+
| binary       |
+-----+
1 row in set
```

MySQL BIT示例

以下语句创建一个名为 `working_calendar` 的新表，其中的列类型和宽度指定为 `BIT(7)`：

```
USE testdb;

CREATE TABLE working_calendar(
    y INT,
    w INT,
    days BIT(7),
    PRIMARY KEY(y,w)
);
```

`days` 列中的值表示工作日或休息日，即 `1` 表示工作日，`0` 表示休息日。

假设 2017 年第一周的星期六和星期五不是工作日，可以在 `working_calendar` 表中插入一行，如下所示：

```
INSERT INTO working_calendar(y,w,days)
VALUES(2017,1,B'1111100');
```

以下查询从 `working_calendar` 表检索数据，结果如下：

```
SELECT
    y, w , days
FROM
    working_calendar;
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| y    | w   | days |
+-----+-----+
| 2017 | 1   | 1111100 |
+-----+-----+
1 row in set
```

如上所见，`days` 列中的位值被转换成一个整数。要将其表示为位值，请使用 `BIN` 函数：

```
SELECT
  y, w , bin(days)
FROM
  working_calendar;
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| y    | w   | bin(days) |
+-----+-----+
| 2017 | 1   | 1111100 |
+-----+-----+
1 row in set
```

如果将值插入到长度小于 `m` 位的 `BIT(m)` 列中，MySQL 将在位值的左侧使用零填充。

假设第二周的第一天关闭，可以向 `days` 列中插入 `01111100` 值。但是，`111100` 的值也可以工作，因为 MySQL 将使用零填充左侧。

```
INSERT INTO working_calendar(y,w,days)
VALUES(2017,2,B'111100');
```

要查看数据，使用与上述相同的查询：

```

SELECT
    y, w , bin(days)
FROM
    working_calendar;

```

执行上面查询语句，得到以下结果 -

y	w	bin(days)
2017	1	1111100
2017	2	111100

2 rows in set

如您所见，MySQL返回结果之前删除了前导零。要正确显示可以使用 LPAD 函数：

```

SELECT
    y, w , lpad(bin(days),7,'0')
FROM
    working_calendar;

```

执行上面查询语句，得到以下结果 -

y	w	lpad(bin(days),7,'0')
2017	1	1111100
2017	2	0111100

2 rows in set

现在工作正常了。

在本教程中，您已经学会了如何使用MySQL BIT 类型来存储位值。

本教程将向您展示如何使用MySQL `BOOLEAN` 数据类型来存储布尔值：`true` 和 `false`。

MySQL BOOLEAN数据类型简介

MySQL没有内置的布尔类型。但是它使用`TINYINT(1)`。为了更方便，MySQL提供 `BOOLEAN` 或 `BOOL` 作为 `TINYINT(1)` 的同义词。

在MySQL中，`0` 被认为是 `false`，非零值被认为是 `true`。要使用布尔文本，可以使用常量 `TRUE` 和 `FALSE` 来分别计算为 `1` 和 `0`。请参阅以下示例：

```
SELECT true, false, TRUE, FALSE, True, False;
-- 1 0 1 0 1 0
```

执行上面代码，得到以下结果 -

```
mysql> SELECT true, false, TRUE, FALSE, True, False;
+-----+-----+-----+-----+-----+
| TRUE | FALSE | TRUE | FALSE | TRUE | FALSE |
+-----+-----+-----+-----+-----+
|    1 |     0 |    1 |     0 |    1 |     0 |
+-----+-----+-----+-----+-----+
1 row in set
```

MySQL BOOLEAN示例

MySQL将布尔值作为整数存储在表中。为了演示，让我们来看下面的 `tasks` 表：

```
USE testdb;

CREATE TABLE tasks (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    completed BOOLEAN
);
```

上面创建表语句中，即使将 `completed` 列指定为 `BOOLEAN` 类型，当显示表定义时，它是却是 `TINYINT(1)` 类型，如下所示：

```
DESCRIBE tasks;
```

以下语句向 `tasks` 表中插入 2 行数据：

```
INSERT INTO tasks(title,completed)
VALUES('Master MySQL Boolean type',true),
      ('Design database table',false);
```

在将数据保存到布尔列之前，MySQL 将其转换为 `1` 或 `0`，以下查询从 `tasks` 表中检索数据：

```
SELECT
    id, title, completed
FROM
    tasks;
```

id	title	completed
1	Master MySQL Boolean type	1
2	Design database table	0

2 rows in set

如上所见，`true` 和 `false` 分别被转换为 `1` 和 `0`。

因为 `Boolean` 类型是 `TINYINT(1)` 的同义词，所以可以在布尔列中插入 `1` 和 `0` 以外的值。如下示例：

```
INSERT INTO tasks(title,completed)
VALUES('Test Boolean with a number',2);
```

上面语句，工作正常~，查询 `tasks` 表中的数据，如下所示 -

```
mysql> SELECT
    id, title, completed
FROM
    tasks;
+----+-----+-----+
| id | title           | completed |
+----+-----+-----+
| 1  | Master MySQL Boolean type | 1 |
| 2  | Design database table   | 0 |
| 3  | Test Boolean with a number | 2 |
+----+-----+-----+
3 rows in set
```

如果要将结果输出为 `true` 和 `false`，可以使用 `IF` 函数，如下所示：

```
SELECT
    id,
    title,
    IF(completed, 'true', 'false') completed
FROM
    tasks;
```

执行上面查询语句，得到结果如下所示 -

```
+----+-----+-----+
| id | title           | completed |
+----+-----+-----+
| 1  | Master MySQL Boolean type | true |
| 2  | Design database table   | false |
| 3  | Test Boolean with a number | true |
+----+-----+-----+
3 rows in set
```

MySQL BOOLEAN 运算符

要在 `tasks` 表中获取所有完成的任务，可以执行以下查询：

```

SELECT
    id, title, completed
FROM
    tasks
WHERE
    completed = TRUE;

```

执行上面查询语句，得到结果如下所示 -

id	title	completed
1	Master MySQL Boolean type	1

1 row in set

如您所见，它只返回 `completed` 列的值为 `1` 的任务。要解决它，必须使用 `IS` 运算符：

```

SELECT
    id, title, completed
FROM
    tasks
WHERE
    completed IS TRUE;

```

执行上面查询语句，得到结果如下所示 -

id	title	completed
1	Master MySQL Boolean type	1
3	Test Boolean with a number	2

2 rows in set

在这个例子中，我们使用 `IS` 运算符来测试一个与布尔值的值。

要获得待处理(未完成)的任务，请使用 `IS FALSE` 或 `IS NOT TRUE`，如下所示：

```
SELECT
    id, title, completed
FROM
    tasks
WHERE
    completed IS NOT TRUE;
```

执行上面查询语句，得到结果如下所示 -

id	title	completed
2	Design database table	0

1 row in set

在本教程中，您已经学习了如何使用MySQL `BOOLEAN` 数据类型(它是 `TINYINT(1)` 的同义词)，以及如何操作布尔值。

在本教程中，您将了解MySQL `CHAR` 数据类型以及如何将其应用于数据库表设计。

MySQL CHAR数据类型简介

`CHAR` 数据类型是MySQL中固定长度的字符类型。我们经常声明 `CHAR` 类型，其长度指定要存储的最大字符数。例如，`CHAR(20)` 最多可以容纳 20 个字符。

如果要存储的数据是固定大小，则应使用 `CHAR` 数据类型。在这种情况下，与 `VARCHAR` 相比，您将获得更好的性能。

`CHAR` 数据类型的长度可以是从 0 到 255 的任何值。当存储 `CHAR` 值时，MySQL 将其值与空格填充到声明的长度。

当查询 `CHAR` 值时，MySQL 会删除尾部的空格。

请注意，如果启用 `PAD_CHAR_TO_FULL_LENGTH` SQL 模式，MySQL 将不会删除尾随空格。

以下语句创建具有 `CHAR` 列的表。

```
USE testdb;

CREATE TABLE mysql_char_test (
    status CHAR(3)
);
```

`status` 列使用 `CHAR` 数据类型，并有指定最多可容纳 3 个字符。

现在，使用以下语句向 `mysql_char_test` 表中插入 2 行数据。

```
INSERT INTO mysql_char_test(status)
VALUES('Yes'), ('No');
```

下面我们使用 `length` 函数来获取每个 `CHAR` 值的长度。

```
SELECT
    status, LENGTH(status)
FROM
    mysql_char_test;
```

执行上面查询语句，得到以下结果 -

status	LENGTH(status)
Yes	3
No	2

2 rows in set

以下语句将插入带有前导和尾随空格的 CHAR 值。

```
INSERT INTO mysql_char_test(status)
VALUES(' Y ');
```

但是，当我们检索该值时，MySQL会删除尾随空格。

```
SELECT
    status, LENGTH(status)
FROM
    mysql_char_test;
```

执行上面查询语句，得到以下结果 -

status	LENGTH(status)
Yes	3
No	2
Y	2

3 rows in set

比较MySQL CHAR值

存储或比较 `CHAR` 值时，MySQL 使用分配给列的字符集排序规则。

使用比较运算符比较 `CHAR` 值时，MySQL 不会考虑尾随空格，例如：`=`，`<>`，`>`，`<` 等等。

请注意，当您使用 `CHAR` 值进行模式匹配时，`LIKE` 运算符会考虑尾随空格。

在前面的例子中，我们将值 `Y` 的前面和后面都加空格存储。但是，当执行以下查询时：

```
mysql> SELECT
    *
FROM
    mysql_char_test
WHERE
    status = 'Y';
Empty set
```

MySQL 没有返回任何行记录，因为它不考虑尾随空格。要与“`Y`”匹配，需要删除尾随空格：

```
SELECT
    *
FROM
    mysql_char_test
WHERE
    status = ' Y';
```

执行上面查询语句，得到以下结果 -

```
+-----+
| status |
+-----+
| Y      |
+-----+
1 row in set
```

MySQL CHAR和UNIQUE索引

如果 `CHAR` 列具有 `UNIQUE` 索引，并且插入的值有多个尾随空格不同的值，则 MySQL 将拒绝因重复键错误而要求您进行的更改。

请参见以下示例

首先，为 `mysql_char_test` 表的 `status` 列创建唯一的索引。

```
CREATE UNIQUE INDEX uidx_status ON mysql_char_test(status);
```

其次，在 `mysql_char_test` 表中插入一行。

```
INSERT INTO mysql_char_test(status)
VALUES('N');
```

```
Error Code: 1062. Duplicate entry 'N' for key 'uidx_status'
```

在本教程中，我们向您介绍了 MySQL `CHAR` 数据类型及其功能。现在，相信您应该很好地了解 `CHAR` 数据类型，以将其应用于数据库设计。

本教程将向您介绍MySQL `VARCHAR` 数据类型，并讨论 `VARCHAR` 的一些重要功能。

MySQL VARCHAR数据类型简介

MySQL `VARCHAR` 是可变长度的字符串，其长度可以达到 65,535 个字符。

MySQL将 `VARCHAR` 值作为 1 字节或 2 字节长度前缀加上实际数据。

长度前缀指定值的字节数。如果列需要少于 255 个字节，则长度前缀为 1 个字节。如果列需要超过 255 个字节，长度前缀是两个长度字节。

但是，最大长度受到最大行大小(65,535 字节)和所使用的字符集的限制。这意味着所有列的总长度应该小于 65,535 字节。

下面我们来看一个例子。

创建一个新的表，它有两列 `s1` 和 `s2`，长度分别为 32765 (长度前缀为 +2) 和 32766(+2)。注意， $32765 + 2 + 32766 + 2 = 65535$ ，这是最大行大小。

```
USE testdb;

CREATE TABLE IF NOT EXISTS varchar_test (
    s1 VARCHAR(32765) NOT NULL,
    s2 VARCHAR(32766) NOT NULL
) CHARACTER SET 'latin1' COLLATE LATIN1_DANISH_CI;
```

该语句成功创建了表。但是，如果我们将 `s1` 列的长度增加 1。

```
USE testdb;

CREATE TABLE IF NOT EXISTS varchar_test_2 (
    s1 VARCHAR(32766) NOT NULL, -- error
    s2 VARCHAR(32766) NOT NULL
) CHARACTER SET 'latin1' COLLATE LATIN1_DANISH_CI;
```

MySQL将发出错误消息：

```
Error Code: 1118. Row size too large. The maximum row size for the used table type, not counting BLOBs, is 65535. This includes storage overhead, check the manual. You have to change some columns to TEXT or BLOBs 0.000 sec
```

如上所示，行长度太大，所以创建语句失败。

如果插入长度大于 `VARCHAR` 列长度的字符串，MySQL 将发出错误。请考虑以下示例：

```
USE testdb;

CREATE TABLE items (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(3)
);

INSERT INTO items(title)
VALUES('ABCD');
```

在这个例子中，MySQL 发出以下错误消息：

```
1406 - Data too long for column 'title' at row 1
```

MySQL VARCHAR 和 空格

当 MySQL 存储 `VARCHAR` 值时，MySQL 不会占用空间。此外，MySQL 在插入或选择 `VARCHAR` 值时保留尾随空格。请参阅以下示例：

```
INSERT INTO items(title)
VALUES('AB ');
```

查询插入字符串值的长度，如下所示 -

```
SELECT
    id, title, length(title)
FROM
    items;

+----+-----+-----+
| id | title | length(title) |
+----+-----+-----+
| 1  | AB   | 3   |
+----+-----+-----+
1 row in set
```

但是，当插入包含导致列长度超过的尾随空格的 VARCHAR 值时，MySQL 将截断尾随空格。此外，MySQL 发出警告。我们来看下面的例子：

```
INSERT INTO items(title)
VALUES('ABC ');
```

此语句将长度为 4 的字符串插入标题 title 列。MySQL 仍然插入字符串，但是在插入该值之前会截断尾随空格。

```
row(s) affected, 1 warning(s): 1265 Data truncated for column 'title' at row 1
```

可以使用以下查询来验证它：

```
SELECT  
    title, LENGTH(title)  
FROM  
    items;
```

title	LENGTH(title)
AB	3
ABC	3

2 rows in set

在本教程中，您已经学习了如何使用MySQL `VARCHAR` 数据类型来存储数据库中的可变长度字符串的值。

在本教程中，您将学习如何使用MySQL `TEXT` 在数据库表中存储文本数据。

MySQL `TEXT` 数据类型简介

MySQL `TEXT` 数据类型是除了`CHAR`和`VARCHAR`字符类型，MySQL为我们提供了具有`CHAR`和`VARCHAR`无法实现的更多功能的`TEXT`类型。

`TEXT` 可用于存储可以从 1 字节到 4GB 长度的文本字符串。我们经常在电子商务网站中找到用于在新闻站点存储物品的 `TEXT` 数据类型，如：产品详细描述。

与 `CHAR` 和 `VARCHAR` 不同，您不必在列使用 `TEXT` 类型时指定存储长度。此外，当检索或插入文本数据(如 `CHAR` 和 `VARCHAR`)时，MySQL不会删除或填充空格。

请注意，`TEXT` 数据不存储在数据库服务器的内存中，因此，每当查询 `TEXT` 数据时，MySQL都必须从磁盘读取它，这与 `CHAR` 和 `VARCHAR` 相比要慢得多。

MySQL提供四种`TEXT`类

型：`TINYTEXT`，`TEXT`，`MEDIUMTEXT` 和 `LONGTEXT`。

下面显示每个 `TEXT` 类型的大小，假设我们使用一个字符集，该字符集需要 1 个字节来存储字符。

TINYTEXT - 1个字节(255个字符)

`TINYTEXT` 可以存储的最大字符是 255 ($2^8 = 256$ ，1 字节开销)。

需要少于 255 个字符的列应该使用 `TINYTEXT` 类型，长度不一致，不需要排序，例如：博文摘录，文章摘要等。

请参阅以下示例：

```
CREATE TABLE articles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    summary TINYTEXT
);
```

在本示例中，我们创建了一个名为 `articles` 的新表，该表具有数据类型为 `TINYTEXT` 的 `summary` 列。

TEXT - 64KB(65,535个字符)

`TEXT` 数据类型最多可容纳 `64KB`，相当于 $65535 (2^{16} - 1)$ 个字符。
`TEXT` 还需要 2 字节开销。

文本可以容纳文章的正文。请考虑以下示例：

```
ALTER TABLE articles
ADD COLUMN body TEXT NOT NULL
AFTER summary;
```

在本示例中，我们使用 `ALTER TABLE` 语句将具有 `TEXT` 数据类型的 `body` 列添加到 `articles` 表。

MEDIUMTEXT - 16MB(16,777,215个字符)

`MEDIUMTEXT` 最多可容纳 `16MB` 的文本数据，相当于 `16,777,215` 个字符。它需要 3 字节开销。`MEDIUMTEXT` 可用于存储相当大的文本数据，如书籍文本，白皮书等。例如：

```
USE testdb;

CREATE TABLE whitepapers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    body MEDIUMTEXT NOT NULL,
    published_on DATE NOT NULL
);
```

LONGTEXT - 4GB(4,294,967,295个字符)

`LONGTEXT` 可以存储高达 `4GB` 的文本数据，这是非常巨大的。它需要 4 字节开销。

在本教程中，您已经学会了如何使用各种MySQL `TEXT` 数据类型来存储数据库表中的文本。

在本教程中，我们将向您介绍MySQL DATE 数据类型，并演示一些有用的日期函数来有效处理日期数据。

MySQL DATE数据类型简介

MySQL DATE 是用于管理日期值的五种时间[数据类型](#)之一。MySQL使用 yyyy-mm-dd 格式存储日期值。此格式是固定的，不可能更改它。

例如，您可能更喜欢使用 mm-dd-yyyy 格式，但是遗憾，不能直接使用。一个代替的办法：遵循标准日期格式，并使用[DATE_FORMAT](#)函数按所需格式来格式化日期。

MySQL使用 3 个字节来存储 DATE 值。 DATE 值的范围为 1000-01-01 到 9999-12-31 。如果要存储超出此范围的日期值，则需要使用非时间数据类型，例如整数，例如使用三列，分别存储年，月和日的数据。还需要创建[存储函数](#)来模拟MySQL提供的内置日期函数，这是不推荐的。

当严格模式被禁用时，MySQL将任何无效日期(例如 2015-02-30)转换为零日期值 0000-00-00 。

MySQL日期值为两位数年份

MySQL使用四位数字存储日期值的年份。如果您使用两位数的年份值，MySQL仍会接受以下规则：

- 年份值在 00-69 范围内转换为 2000-2069 。
- 70-99 的年值被转换为 1970 - 1999 年。

但是，具有两位数字的日期值是不明确的，因此您应避免使用它。

现在，让我们来看下面的例子。

首先，创建一个名为 people 表，其生日(birth_date)列使用 DATE 数据类型。

```
USE testdb;

CREATE TABLE people (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    birth_date DATE NOT NULL
);
```

接下来，将一行插入到 `people` 表中。

```
INSERT INTO people(first_name, last_name, birth_date)
VALUES('Max', 'Su', '1992-10-11');
```

然后，查询 `people` 表中的数据，如下所示 -

```
SELECT
    first_name,
    last_name,
    birth_date
FROM
    people;
```

执行上面查询语句，得到以下结果 -

first_name	last_name	birth_date
Max	Su	1992-10-11

1 row in set

之后，使用两位数的年份格式将数据插入到 `people` 表中。

```
INSERT INTO people(first_name, last_name, birth_date)
VALUES('Jack', 'Daniel', '01-09-01'),
      ('Lily', 'Bush', '80-09-01');
```

在第一行，我们使用 01 (范围在 00-69)作为年份，所以MySQL将其转换为 2001 年。在第二行，我们使用 80 (范围 70-99)作为年份，MySQL将其转换为 1980 年。

最后，从 `people` 表查询数据，以检查数据是否根据转换规则进行转换。

```
SELECT
    first_name,
    last_name,
    birth_date
FROM
    people;
```

执行上面查询语句，得到以下结果 -

first_name	last_name	birth_date
Max	Su	1992 10 11
Jack	Daniel	2001 09 01
Lily	Bush	1980 09 01

3 rows in set

MySQL DATE 函数

MySQL提供了许多有用的日期功能，可以有效地操作日期。

要获取当前日期和时间，请使用[NOW\(\)](#)函数。

```
SELECT NOW() as cur_datetime;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT NOW() as cur_datetime;
+-----+
| cur_datetime |
+-----+
| 2017-07-25 21:51:54 |
+-----+
1 row in set
```

要获取**DATETIME**值的日期部分，可以使用 **DATE()** 函数。

```
SELECT DATE(NOW());
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT DATE(NOW());
+-----+
| DATE(NOW()) |
+-----+
| 2017-07-25 |
+-----+
1 row in set
```

要获取当前的系统日期，可以使用**CURDATE()**函数，如下所示：

```
SELECT CURDATE();
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT CURDATE();
+-----+
| CURDATE() |
+-----+
| 2017-07-25 |
+-----+
1 row in set
```

要格式化日期值，可以使用DATE_FORMAT函数。以下语句使用日期格式模式 %m/%d/%Y ，格式化日期为： mm/dd/yyyy :

```
SELECT DATE_FORMAT(CURDATE(), '%m/%d/%Y') today;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT DATE_FORMAT(CURDATE(), '%m/%d/%Y') today;
+-----+
| today |
+-----+
| 07/25/2017 |
+-----+
1 row in set
```

要计算两个日期值之间的天数，可以使用DATEDIFF函数，如下所示：

```
SELECT DATEDIFF('2015-11-04', '2014-11-04') days;
```

执行上面查询语句，得到以下结果 -

要添加几天，几周，几个月，几年等到一个日期值，可以使用DATE_ADD函数：

```
SELECT
    '2018-01-01' start,
    DATE_ADD('2018-01-01', INTERVAL 1 DAY) 'one day later',
    DATE_ADD('2018-01-01', INTERVAL 1 WEEK) 'one week later',
    DATE_ADD('2018-01-01', INTERVAL 1 MONTH) 'one month later',
    DATE_ADD('2018-01-01', INTERVAL 1 YEAR) 'one year later';
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+-----+
+-----+
| start | one day later | one week later | one month later |
| one year later |
+-----+
| 2018-01-01 | 2018-01-02 | 2018-01-08 | 2018-02-01 |
| 2019-01-01 |
+-----+
+-----+
1 row in set
```

类似地，可以使用DATE_SUB函数从日期中减去间隔值：

```
SELECT
    '2018-01-01' start,
    DATE_SUB('2018-01-01', INTERVAL 1 DAY) 'one day before',
    DATE_SUB('2018-01-01', INTERVAL 1 WEEK) 'one week before',
    DATE_SUB('2018-01-01', INTERVAL 1 MONTH) 'one month before',
    DATE_SUB('2018-01-01', INTERVAL 1 YEAR) 'one year before';
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+-----+
+-----+
| start | one day before | one week before | one month before |
| one year before |
+-----+
| 2018-01-01 | 2017-12-31 | 2017-12-25 | 2017-12-01 |
| 2017-01-01 |
+-----+
+-----+
1 row in set
```

如果要获取日期值的日期，月份，季度和年份，可以使用相应的函数：DAY，MONTH，QUARTER 和YEAR，如下所示：

```
SELECT DAY('2018-12-31') day,
       MONTH('2018-12-31') month,
      QUARTER('2018-12-31') quarter,
        YEAR('2018-12-31') year;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT DAY('2018-12-31') day,
           MONTH('2018-12-31') month,
          QUARTER('2018-12-31') quarter,
            YEAR('2018-12-31') year;
+-----+-----+-----+
| day | month | quarter | year |
+-----+-----+-----+
| 31  |    12 |        4 | 2018 |
+-----+-----+-----+
1 row in set
```

获得周信息周相关功能。例如，`WEEK`函数返回周数，`WEEKDAY` 函数返回工作日索引，`WEEKOFYEAR` 函数返回周日历。

```
SELECT
  WEEKDAY('2018-12-31') weekday,
  WEEK('2018-12-31') week,
  WEEKOFYEAR('2018-12-31') weekofyear;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
           WEEKDAY('2018-12-31') weekday,
             WEEK('2018-12-31') week,
            WEEKOFYEAR('2018-12-31') weekofyear;
+-----+-----+-----+
| weekday | week | weekofyear |
+-----+-----+-----+
|      0  |   52 |         1 |
+-----+-----+-----+
1 row in set
```

如果没有传递第二个参数，或者如果传递参数为 0，则 week 函数将返回带有零的索引的周数。如果传递参数为 1，则将返回 1 索引的周数。

```
SELECT  
WEEKDAY('2018-12-31') weekday,  
WEEK('2018-12-31',1) week,  
WEEKOFYEAR('2018-12-31') weekofyear;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT  
WEEKDAY('2018-12-31') weekday,  
WEEK('2018-12-31',1) week,  
WEEKOFYEAR('2018-12-31') weekofyear;  
+-----+-----+-----+  
| weekday | week | weekofyear |  
+-----+-----+-----+  
|      0 |    53 |        1 |  
+-----+-----+-----+  
1 row in set
```

在本教程中，您已经了解并学习了MySQL DATE 数据类型以及如何使用一些有用的日期函数来操作日期值。

在本教程中，我们将向您介绍MySQL `TIME` 数据类型，并显示有用的时间函数来有效地处理时间数据。

MySQL `TIME` 数据类型简介

MySQL 使用 `HH:MM:SS` 格式来查询和显示代表一天中的时间值(在 24 小时内)。要表示两个事件之间的时间间隔，MySQL 使用大于 24 小时的 `HHH:MM:SS` 格式。

要定义 `TIME` 数据类型的列，请使用以下语法：

```
column_name TIME;
```

例如，以下代码片段定义了一个名为 `start_at` 的列，其中包含 `TIME` 数据类型。

```
start_at TIME;
```

`TIME` 值范围为 `-838:59:59` 至 `838:59:59`。此外，`TIME` 值可以具有高达微秒精度(6位数)的小数秒部分。要使用小数秒精度部分定义数据类型为 `TIME` 的列，请使用以下语法：

```
column_name TIME(N);
```

`N` 是表示小数部分的整数值，最多 6 位数。

以下代码片段定义了 `TIME` 数据类型的列，其中包含 3 位数的小数秒。

```
begin_at TIME(3);
```

`TIME` 值需要 3 个字节进行存储。如果 `TIME` 值包括分数秒精度，则会根据小数秒精度的位数获取额外的字节。下表说明了小数秒精度所需的存储空间。

分数秒精度	存储(字节)
0	0
1, 2	1
3, 4	2
5, 6	3

例如，`TIME` 和 `TIME(0)` 需要 3 个字节。`TIME(1)` 和 `TIME(2)` 需要 4 个字节($3 + 1$)；`TIME(3)` 和 `TIME(6)` 分别需要 5 和 6 个字节。

MySQL TIME数据类型示例

让我们来看一下在表中对列使用 `TIME` 数据类型的例子。

首先，创建一个名为 `tests` 的新表，其中包含四个列：`id`，`name`，`start_at` 和 `end_at`。`start_at` 和 `end_at` 列的数据类型为 `TIME`。

```
USE testdb;

CREATE TABLE tests (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    start_at TIME,
    end_at TIME
);
```

第二步，在 `tests` 表中插入一行。

```
INSERT INTO tests(name,start_at,end_at)
VALUES('Test 1', '08:00:00','10:00:00');
```

第三步，从 `tests` 表查询数据，如下语句：

```
SELECT
    name, start_at, end_at
FROM
    tests;
```

执行上面查询语句，得到以下结果 -

请注意，我们使用“`HH:MM:SS`”作为`INSERT`语句中的文字时间值。下面来看看 MySQL 可以识别的所有有效的时间文字。

MySQL TIME 文字

除了我们之前提到的“`HH:MM:SS`”格式之外，MySQL 还可以识别各种时间格式。

MySQL 允许使用“`HHMMSS`”格式，而不使用分隔符(`:`)表示时间值。例如' `08:30:00` ‘和’ `10:15:00` ‘可以重写为’ `083000` ‘和’ `101500` ‘。

```
INSERT INTO tests(name,start_at,end_at)
VALUES('Test 2','083000','101500');
```

但是，`108000` 不是有效的时间值，因为 `80` 不代表正确的分钟。在这种情况下，如果您尝试在表中插入无效的时间值，MySQL 会引发错误。

```
INSERT INTO tests(name,start_at,end_at)
VALUES('Test invalid','083000','108000');
```

执行上述语句后，MySQL 发出以下错误消息。

```
Error Code: 1292. Incorrect time value: '108000' for column 'end_at' at row 1
```

除了字符串格式之外，MySQL接受 `HHMMSS` 作为代表时间值的数字。也可以使用 `SS`，`MMSS`。例如，可以使用 `082000`，而不是使用 `'082000'`，如下所示：

```
INSERT INTO tests(name,start_at,end_at)
VALUES('Test 3',082000,102000);
```

对于时间间隔，您可以使用 `'D HH:MM:SS'` 格式，其中 `D` 代表天数从 `0` 到 `34` 的范围。更灵活的语法是 `'HH:MM'`，`'D HH:MM'`，`'D HH'` 或 `'SS'`。

如果使用分隔符 `:`，可以使用 1 位数字表示小时，分钟或秒。例如，可以使用 `9:5:0` 而不是 `'09:05:00'`。

```
INSERT INTO tests(name,start_at,end_at)
VALUES('Test 4','9:5:0',100500);
```

MySQL TIME函数

MySQL提供了几个有用的时间函数来处理 `TIME` 数据。

获取当前时间

要获取数据库服务器的当前时间，请使用 `CURRENT_TIME` 函数。根据使用该函数的上下文，`CURRENT_TIME` 函数以字符串(`'HH:MM:SS'`)或数值(`HHMMSS`)返回当前时间值。

以下语句说明了字符串和数字上下文中的 `CURRENT_TIME` 函数：

```
SELECT
  CURRENT_TIME() AS string_now,
  CURRENT_TIME() + 0 AS numeric_now;
```

执行上面查询语句，得到以下结果 -

```
+-----+
| string_now | numeric_now |
+-----+
| 23:04:53 |      230453 |
+-----+
1 row in set
```

从**TIME**值添加和减去时间

要将 **TIME** 值添加到另一个 **TIME** 值，请使用 **ADDTIME** 函数。要从另一个 **TIME** 值中减去 **TIME** 值，可以使用 **SUBTIME** 函数。

以下语句从当前时间起减去 2 小时 30 分钟。

```
SELECT
  CURRENT_TIME(),
  ADDTIME(CURRENT_TIME(), '023000'),
  SUBTIME(CURRENT_TIME(), '023000');
```

执行上面查询语句，得到以下结果 -

```
+-----+
| CURRENT_TIME() | ADDTIME(CURRENT_TIME(), '023000') | SUBTIME(CUR
RENT_TIME(), '023000) |
+-----+
| 23:05:03 | 25:35:03 | 20:35:03
+-----+
1 row in set
```

此外，可以使用 **TIMEDIFF()** 函数来获取两个 **TIME** 值之间的差异。

```

SELECT
    TIMEDIFF(end_at, start_at)
FROM
    tests;

```

执行上面查询语句，得到以下结果 -

```

mysql> SELECT
    TIMEDIFF(end_at, start_at)
FROM
    tests;
+-----+
| TIMEDIFF(end_at, start_at) |
+-----+
| 02:00:00
| 01:45:00
| 02:00:00
| 01:00:00
+-----+
4 rows in set

```

格式化MySQL TIME值

虽然MySQL在检索和显示 TIME 值时使用“ HH:MM:SS ”，但可以使用 TIME_FORMAT 函数以推荐的方式显示 TIME 值。

TIME_FORMAT 函数类似于 DATE_FORMAT 函数，除了 TIME_FORMAT 函数仅用于格式化 TIME 值其它均差不多。

请参见以下一个示例 -

```

SELECT
    name,
    TIME_FORMAT(start_at, '%h:%i %p') start_at,
    TIME_FORMAT(end_at, '%h:%i %p') end_at
FROM
    tests;

```

执行上面查询语句，得到以下结果 -

```
+-----+-----+-----+
| name | start_at | end_at |
+-----+-----+-----+
| Test 1 | 08:00 AM | 10:00 AM |
| Test 2 | 08:30 AM | 10:15 AM |
| Test 3 | 08:20 AM | 10:20 AM |
| Test 4 | 09:05 AM | 10:05 AM |
+-----+-----+-----+
4 rows in set
```

在上面的时间格式字符串中：

- `%h` 表示从 0 到 12 的两位数小时数值。
- `%i` 表示从 0 到 60 的两位数分钟数值。
- `%p` 表示 AM 或 PM，也就是表示上午或下午。

从**TIME**值提取小时，分钟和秒

要从 `TIME` 值中提取小时，分和秒，可以使用 `HOUR`，`MINUTE` 和 `SECOND` 函数，如下所示：

	start_at	h	m	s
▶	08:00:00	8	0	0
	08:30:00	8	30	0
	08:20:00	8	20	0
	09:05:00	9	5	0

获取**UTC**时间值

要获取 `UTC` 时间，请使用 `UTC_TIME` 函数，如下所示：

```
SELECT
  CURRENT_TIME(),
  UTC_TIME();
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| CURRENT_TIME() | UTC_TIME() |
+-----+-----+
| 23:05:41      | 15:05:41   |
+-----+-----+
1 row in set
```

在本教程中，我们已经介绍了有关MySQL `TIME` 数据类型和一些常用的时间函数来处理 `TIME` 值。

在本教程中，您将了解MySQL `DATETIME` 数据类型以及如何使用一些方便的函数来有效地操作 `DATETIME` 类型数据。

MySQL DATETIME数据类型简介

MySQL `DATETIME` 存储包含日期和时间的值。当您从 `DATETIME` 列查询数据时，MySQL会以以下格式显示 `DATETIME` 值：

`YYYY-MM-DD HH:MM:SS`

默认情况下，`DATETIME` 的值范围为 `1000-01-01 00:00:00` 至 `9999-12-31 23:59:59`。

`DATETIME` 值使用 5 个字节进行存储。另外，`DATETIME` 值可以包括格式为 `YYYY-MM-DD HH:MM:SS [.fraction]` 例如：`2017-12-20 10:01:00.999999` 的尾数有小数秒。当包含小数秒精度时，`DATETIME` 值需要更多存储，如下表所示：

分数秒精度	存储(字节)
0	0
1, 2	1
3, 4	2
5, 6	3

例如，`2017-12-20 10:01:00.999999` 需要 8 个字节，`2015-12-20 10:01:00` 需要 5 个字节，3 个字节为 `.999999`，而 `2017-12-20 10:01:00.9` 只需要 6 个字节，小数秒精度为 1 字节。

请注意，在MySQL 5.6.4之前，`DATETIME` 值需要 8 字节存储而不是 5 个字节。

MySQL DATETIME与TIMESTAMP类型

MySQL提供了另一种类似于 `DATETIME`，叫作`TIMESTAMP`的时间数据类型。

`TIMESTAMP` 需要 4 个字节，而 `DATETIME` 需要 5 个字节。

`TIMESTAMP` 和 `DATETIME` 都需要额外的字节，用于分数秒精度。

`TIMESTAMP` 值范围从 `1970-01-01 00:00:01 UTC` 到 `2038-01-19 03:14:07 UTC`。如果要存储超过 `2038` 的时间值，则应使用 `DATETIME` 而不是 `TIMESTAMP`。

MySQL 将 `TIMESTAMP` 存储在 `UTC` (有时区)值中。但是，MySQL 存储 `DATETIME` 值是没有时区的。下面来看看下面的例子。

首先，将当前连接的时区设置为 `+00:00`。

接下来，创建一个名为 `timestamp_n_datetime` 的表，它由两列组成：`ts` 和 `dt`，这两列分别使用 `TIMESTAMP` 和 `DATETIME` 类型，如以下语句 -

```
USE testdb;

CREATE TABLE timestamp_n_datetime (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ts TIMESTAMP,
    dt DATETIME
);
```

然后，将当前日期和时间插入到 `timestamp_n_datetime` 表的 `ts` 和 `dt` 列中，如下语句所示 -

```
INSERT INTO timestamp_n_datetime(ts,dt)
VALUES(NOW(),NOW());
```

之后，从 `timestamp_n_datetime` 表查询数据，如下语句所示 -

```
SELECT
    ts,
    dt
FROM
    timestamp_n_datetime;
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| ts      | dt      |
+-----+-----+
| 2017-07-26 00:38:14 | 2017-07-26 00:38:14 |
+-----+-----+
1 row in set
```

DATETIME 和 TIMESTAMP 列中的两个值相同。

最后，将连接的时区设置为 +03:00 ，再次从 timestamp_n_datetime 表查询数据。

```
SET time_zone = '+03:00';

SELECT
    ts,
    dt
FROM
    timestamp_n_datetime;
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| ts      | dt      |
+-----+-----+
| 2017-07-25 19:38:14 | 2017-07-26 00:38:14 |
+-----+-----+
1 row in set
```

可以看到， ts 列为 TIMESTAMP 数据类型的值变了。这是因为在更改时区时， TIMESTAMP 列以 UTC 为单位存储日期和时间值，根据新时区调整 TIMESTAMP 列的值。

这意味着如果使用 TIMESTAMP 数据来存储日期和时间值，则在将数据库移动到位于不同时区的服务器时时间的值可能不一样，所以应该认真考虑这个问题。

MySQL DATETIME 函数

以下语句使用 NOW() 函数将变量 @dt 设置为当前日期和时间。

```
SET @dt = NOW();
```

要查询 `@dt` 变量的值，请使用以下 `SELECT` 语句：

```
SELECT @dt;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT @dt;
+-----+
| @dt |
+-----+
| 2017-07-25 19:41:14 |
+-----+
1 row in set
```

MySQL DATE 函数

要从 `DATETIME` 值提取日期部分，请使用 `DATE` 函数，如下所示：

```
mysql> SELECT DATE(@dt);
+-----+
| DATE(@dt) |
+-----+
| 2017-07-25 |
+-----+
1 row in set
```

如果希望根据日期查询数据，但是列中存储的数据是基于日期和时间，则此功能非常有用。

下面来看看下面的例子。

```
USE testdb;

CREATE TABLE test_dt (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME
);

INSERT INTO test_dt(created_at)
VALUES('2017-11-05 20:29:36');
```

假设您想知道在 2017-11-05 当天创建的行，请使用以下查询：

```
SELECT
*
FROM
    test_dt
WHERE
    created_at = '2017-11-05';
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
*
FROM
    test_dt
WHERE
    created_at = '2017-11-05';
Empty set
```

它不返回任何行记录。

这是因为 `created_at` 列不仅包含日期，还包含时间。要纠正它，请使用 `DATE` 函数，如下所示：

```

SELECT
    *
FROM
    test_dt
WHERE
    DATE(created_at) = '2017-11-05';

```

执行上面查询语句，得到以下结果 -

```

+-----+
| id | created_at |
+-----+
| 1  | 2017-11-05 20:29:36 |
+-----+
1 row in set

```

它按预期返回一行。如果表有多行，MySQL必须执行全表扫描以查找与条件匹配的行。

MySQL TIME函数

要从 DATETIME 值中提取时间部分，可以使用 TIME 函数，如以下语句所示：

```
SELECT TIME(@dt);
```

执行上面查询语句，得到以下结果 -

```

mysql> SELECT TIME(@dt);
+-----+
| TIME(@dt) |
+-----+
| 19:41:14 |
+-----+
1 row in set

```

MySQL YEAR, QUARTER, MONTH, WEEK, DAY, HOUR，MINUTE和SECOND 函数

要从 `DATETIME` 值获取年，季，月，周，日，小时，分和秒，可以使用以下语句中所示的函数：

```
SET @dt = NOW();
SELECT
    HOUR(@dt),
    MINUTE(@dt),
    SECOND(@dt),
    DAY(@dt),
    WEEK(@dt),
    MONTH(@dt),
    QUARTER(@dt),
    YEAR(@dt);
```

执行上面查询语句，得到以下结果 -

HOUR(@dt)	MINUTE(@dt)	SECOND(@dt)	DAY(@dt)	WEEK(@dt)
MONTH(@dt)	QUARTER(@dt)	YEAR(@dt)		
19	42	56	25	30
7	3	2017		

1 row in set

MySQL DATE_FORMAT 函数

要格式化 `DATETIME` 值，可以使用 `DATE_FORMAT` 函数。例如，以下语句基于 `%H:%i:%s - %w%M%Y` 格式来格式化 `DATETIME` 值：

```
SET @dt = NOW();
SELECT DATE_FORMAT(@dt, '%H:%i:%s - %w %M %Y');
```

执行上面查询语句，得到以下结果 -

```
+-----+  
| DATE_FORMAT(@dt, '%H:%i:%s - %W %M %Y') |  
+-----+  
| 19:43:10 - Tuesday July 2017 |  
+-----+  
1 row in set
```

MySQL DATE_ADD函数

要将间隔添加到 DATETIME 值，请使用DATE_ADD函数，如下所示：

```
SET @dt = NOW();  
SELECT @dt start,  
       DATE_ADD(@dt, INTERVAL 1 SECOND) '1 second later',  
       DATE_ADD(@dt, INTERVAL 1 MINUTE) '1 minute later',  
       DATE_ADD(@dt, INTERVAL 1 HOUR) '1 hour later',  
       DATE_ADD(@dt, INTERVAL 1 DAY) '1 day later',  
       DATE_ADD(@dt, INTERVAL 1 WEEK) '1 week later',  
       DATE_ADD(@dt, INTERVAL 1 MONTH) '1 month later',  
       DATE_ADD(@dt, INTERVAL 1 YEAR) '1 year later';
```

执行上面查询语句，得到以下结果 -

start	1 second later	1 minute later
1 hour later	1 day later	1 week later
1 month later	1 year later	
2017 07 25 19:43:22	2017 07 25 19:43:23	2017 07 25 19:44:22
2017 07 25 20:43:22	2017 07 26 19:43:22	2017 08 01 19:43:
22	2017 08 25 19:43:22	2018 07 25 19:43:22

1 row in set

MySQL DATE_SUB函数

要从 DATETIME 值中减去一个间隔值，可以使用DATE_SUB函数，如下所示：

```
SET @dt = NOW();
SELECT @dt start,
       DATE_SUB(@dt, INTERVAL 1 SECOND) '1 second before',
       DATE_SUB(@dt, INTERVAL 1 MINUTE) '1 minute before',
       DATE_SUB(@dt, INTERVAL 1 HOUR) '1 hour before',
       DATE_SUB(@dt, INTERVAL 1 DAY) '1 day before',
       DATE_SUB(@dt, INTERVAL 1 WEEK) '1 week before',
       DATE_SUB(@dt, INTERVAL 1 MONTH) '1 month before',
       DATE_SUB(@dt, INTERVAL 1 YEAR) '1 year before';
```

执行上面查询语句，得到以下结果 -

start	1 second before	1 minute before
1 hour before	1 day before	1 week before
1 month before	1 year before	

MySQL DATE_DIFF 函数

要计算两个 `DATETIME` 值之间的差值，可以使用 `DATEDIFF` 函数。请注意，`DATEDIFF` 函数仅在计算中考虑 `DATETIME` 值的日期部分。

请参见以下示例。

首先，创建一个名为 `datediff_test` 的表，其中只有一个 `dt` 列，其数据类型为 `DATETIME`。

```
USE testdb;
CREATE TABLE datediff_test (
    dt DATETIME
);
```

其次，将一些行插入到 `datediff_test` 表中。

```
INSERT INTO datediff_test(dt)
VALUES('2017-04-30 07:27:39'),
      ('2017-05-17 22:52:21'),
      ('2017-05-18 01:19:10'),
      ('2017-05-22 14:17:16'),
      ('2017-05-26 03:26:56'),
      ('2017-06-10 04:44:38'),
      ('2017-06-13 13:55:53');
```

第三，使用 `DATEDIFF` 函数将当前日期时间与 `datediff_test` 表的每一行中的值进行比较。

```
SELECT
    dt,
    DATEDIFF(NOW(), dt)
FROM
    datediff_test;
```

执行上面查询语句，得到以下结果 -

dt	DATEDIFF(NOW(), dt)
2017-04-30 07:27:39	86
2017-05-17 22:52:21	69
2017-05-18 01:19:10	68
2017-05-22 14:17:16	64
2017-05-26 03:26:56	60
2017-06-10 04:44:38	45
2017-06-13 13:55:53	42

7 rows in set

在本教程中，您已经了解了MySQL `DATETIME` 数据类型和一些有用的 `DATETIME` 函数。

在本教程中，您将了解MySQL `TIMESTAMP` 和 `TIMESTAMP` 列的功能，如使用时间戳自动初始化和更新。

MySQL TIMESTAMP简介

MySQL `TIMESTAMP` 是一种保存日期和时间组合的时间数据类型。

`TIMESTAMP` 列的格式为 `YYYY-MM-DD HH:MM:SS`，固定为 19 个字符。

`TIMESTAMP` 值的范围从 '`1970-01-01 00:00:01`' UTC 到 '`2038-01-19 03:14:07`' UTC。

当您将 `TIMESTAMP` 值插入到表中时，MySQL 会将其从连接的时区转换为 UTC 后进行存储。当您查询 `TIMESTAMP` 值时，MySQL 会将 UTC 值转换回连接的时区。请注意，对于其他时间数据类型(如 `DATETIME`)，不会进行此转换。

当检索由不同时区中的客户端插入 `TIMESTAMP` 值时，将获得存储数据库中不同的值。只要不更改时区，就可以获得与存储的相同的 `TIMESTAMP` 值。

MySQL TIMESTAMP时区示例

我们来看一个例子来看看 MySQL 如何处理 `TIMESTAMP` 值。

首先，创建一个名为 `test_timestamp` 的新表，该表具有一列：`t1`，其数据类型为 `TIMESTAMP`；

```
USE testdb;
CREATE TABLE IF NOT EXISTS test_timestamp (
    t1 TIMESTAMP
);
```

其次，使用 `SET time_zone` 语句将时区设置为 "`+00:00`" UTC。

```
SET time_zone = '+00:00';
```

第三，将 `TIMESTAMP` 值插入到 `test_timestamp` 表中。

```
INSERT INTO test_timestamp
VALUES('2018-01-01 00:00:01');
```

第四，从 `test_timestamp` 表中查询选择 `TIMESTAMP` 值。

```
SELECT
    t1
FROM
    test_timestamp;
```

t1
2018-01-01 00:00:01

1 row in set

第五，将会话时区设置为不同的时区，以查看从数据库服务器返回的值：

```
SET time_zone = '+03:00';

SELECT t1
FROM test_timestamp;
```

执行上面查询语句，得到以下结果 -

t1
2018-01-01 03:00:01

1 row in set

如上面所见，查询结果集中为调整到新时区的不同时间值。

将 **TIMESTAMP** 列的自动初始化和更新

下面，我们从一个例子开始。以下语句创建一个名为 `categories` 的表：

```
USE testdb;

CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

在类别表中，`created_at` 列是一个 `TIMESTAMP` 列，其默认值设置为 `CURRENT_TIMESTAMP`。以下语句向 `categories` 表中插入一个新行，而不指定 `created_at` 列的值：

```
INSERT INTO categories(name)
VALUES ('A');

SELECT
*
FROM
categories;
```

执行上面查询语句，得到以下结果 -

<code>id</code>	<code>name</code>	<code>created_at</code>
1	A	2017-07-25 21:52:46

1 row in set

可以看到，MySQL 使用时间戳(在插入行时)为 `created_at` 列初始化。

因此，`TIMESTAMP` 列可以自动初始化为指定列插入行的当前时间戳作为一个值。此功能称为自动初始化。

我们将添加一个名为 `updated_at`，数据类型为 `TIMESTAMP` 的新列到 `categories` 表中。

```
ALTER TABLE categories
ADD COLUMN updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;
```

`updated_at` 列的默认值为 `CURRENT_TIMESTAMP`。但是，在 `DEFAULT CURRENT_TIMESTAMP` 子句之后又有一个子句 `ON UPDATE CURRENT_TIMESTAMP`。下面我们来看看它表示什么意思。

以下语句将一个新行插入到 `categories` 表中。

```
INSERT INTO categories(name)
VALUES('B');

SELECT * FROM categories;
```

执行上面查询语句，得到以下结果 -

<code>id</code>	<code>name</code>	<code>created_at</code>	<code>updated_at</code>
1	A	2017-07-25 21:52:46	2017-07-25 21:53:16
2	B	2017-07-25 21:53:25	2017-07-25 21:53:25

2 rows in set

`created_at` 列的默认值是插入行的时间戳。

现在，我们更新 `id=2` 的行的 `name` 列中的值，并从 `categories` 表更新查询数据。

```
UPDATE categories
SET
    name = 'B+'
WHERE
    id = 2;
```

查询更新结果 -

```

SELECT
*
FROM
categories
WHERE
id = 2;

```

执行上面查询语句，得到以下结果 -

<code>id</code>	<code>name</code>	<code>created_at</code>	<code>updated_at</code>
2	B	2017-07-25 21:53:25	2017-07-25 21:53:25

1 row in set

请注意，`updated_at` 列中的值在更新行时自动更改了时间戳。

当行中任何其他列中的值从其当前值更改时，`TIMESTAMP` 列的功能将自动更新为当前时间戳，这种行为称为自动更新。

`updated_at` 列被称为自动更新列。

请注意，如果执行 `UPDATE` 语句以更新 `name` 列的相同值，则 `updated_at` 列将不会更新。

```

UPDATE categories
SET
name = 'B+'
WHERE
id = 2;

```

`updated_at` 列的值保持不变。

有关自动初始化和更新的更多信息，请查看 MySQL 网站上的[时间初始化](#)。

从 MySQL 5.6.5 开始，`DATETIME` 列还具有自动初始化和更新功能。此外，`DEFAULT_CURRENT_TIMESTAMP` 和 `ON UPDATE CURRENT_TIMESTAMP` 可以应用于多个列。

在本教程中，我们向您介绍了MySQL `TIMESTAMP` 数据类型，并向您展示了如何使用 `TIMESTAMP` 列的自动初始化和更新功能。

在本教程中，您将学习如何使用MySQL `JSON` 数据类型将 `JSON` 文档存储在数据库中。

MySQL JSON数据类型简介

自MySQL5.7.8版本以来，MySQL支持原生JSON数据类型。允许使用原生JSON数据类型比以前MySQL版本中所使用JSON文本格式更能有效地存储JSON文档。

MySQL以内部格式存储JSON文档，允许对文档元素的快速读取访问。JSON二进制格式的结构是允许服务器通过键或数组索引直接搜索JSON文档中的值，这非常快。

JSON文档的存储大约与存储 `LONGBLOB` 或 `LONGTEXT` 数据量相同。

要定义数据类型为JSON的列，请使用以下语法：

```
CREATE TABLE table_name (
    ...
    json_column_name JSON,
    ...
);
```

请注意，JSON列不能有默认值。此外，JSON列不能直接编入索引。可以在包含从JSON列中提取的值的[生成列](#)上创建索引。当从JSON列[查询数据](#)时，MySQL优化器将在匹配JSON表达式的虚拟列上查找兼容的索引。

MySQL JSON数据类型示例

假设跟踪访客在网站上的行为。一些访问者可能只是查看页面，而其他访问者可能会查看页面并购买产品。要存储这些信息，我们将[创建一个名为 events 的新表](#)。

```
USE testdb;
CREATE TABLE events(
    id int auto_increment primary key,
    event_name varchar(255),
    visitor varchar(255),
    properties json,
    browser json
);
```

事件表中的每个事件都有一个唯一标识事件的 `id`。事件还有一个 `event_name` 列，例如浏览量，购买等。`visitor` 列用于存储访问者信息。

`properties` 和 `browser` 列是JSON类型。它们用于存储访问者浏览网站的事件属性和浏览器信息(如版本，名称等等)。

我们将一些数据插入到 `events` 表中：

```
INSERT INTO events(event_name, visitor, properties, browser)
VALUES (
    'pageview',
    '1',
    '{ "page": "/" }',
    '{ "name": "Safari", "os": "Mac", "resolution": { "x": 1920, "y": 1080 } }'
),
(
    'pageview',
    '2',
    '{ "page": "/contact" }',
    '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 2560, "y": 1600 } }'
),
(
    'pageview',
    '1',
    '{ "page": "/products" }',
    '{ "name": "Safari", "os": "Mac", "resolution": { "x": 1920, "y": 1080 } }'
),
(
    'purchase',
```

```

    '3',
    '{ "amount": 200 }',
    '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 16
00, "y": 900 } }'
),
(
    'purchase',
    '4',
    '{ "amount": 150 }',
    '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 12
80, "y": 800 } }'
),
(
    'purchase',
    '4',
    '{ "amount": 500 }',
    '{ "name": "Chrome", "os": "Windows", "resolution": { "x": 168
0, "y": 1050 } }'
);

```

要从JSON列中引出值，可以使用列路径运算符(`->`)。

```
SELECT id, browser->'$.name' browser FROM events;
```

此查询返回以下输出：

id	browser
1	"Safari"
2	"Firefox"
3	"Safari"
4	"Firefox"
5	"Firefox"
6	"Chrome"
7	"Safari"

7 rows in set (0.00 sec)

请注意，上面查询语句要在命令中连接到MySQL服务器(`mysql -hlocalhost -uroot -p`)。可以看到 `browser` 列中的数据被引号包围。要删除引号，请使用内联路径运算符(`->>`)，如下所示：

```
SELECT id, browser->>'$.name' browser
FROM events;
```

从以下输出可以看出，引号已被删除：

	<code>id</code>	<code>browser</code>
1		Safari
2		Firefox
3		Safari
4		Firefox
5		Firefox
6		Chrome

6 rows in set (0.00 sec)

要获取浏览器的使用情况，可以使用以下语句：

```
SELECT browser->>'$.name' browser,
       count(browser)
  FROM events
 GROUP BY browser->>'$.name';
```

上面查询语句的输出如下：

<code>browser</code>	<code>count(browser)</code>
Safari	2
Firefox	3
Chrome	1

3 rows in set (0.02 sec)

要计算访问者的总购买量，请使用以下查询：

```
SELECT visitor, SUM(properties->>'$.amount') revenue
FROM events
WHERE properties->>'$.amount' > 0
GROUP BY visitor;
```

上面查询语句的输出如下：

visitor	revenue
3	200
4	650

2 rows in set (0.00 sec)

在本教程中，您已经了解了MySQL JSON 数据类型以及如何使用它来存储数据库中的 JSON 文档。

在本教程中，您将学习如何使用MySQL `ENUM` 数据类型定义存储枚举值的列。

MySQL ENUM数据类型简介

在MySQL中，`ENUM` 是一个字符串对象，其值是从列创建时定义的允许值列表中选择的。

`ENUM` 数据类型提供以下优点：

- 紧凑型数据存储，MySQL `ENUM` 使用数字索引(1, 2, 3, ...)来表示字符串值。
- 可读查询和输出。

要定义 `ENUM` 列，请使用以下语法：

```
CREATE TABLE table_name (
    ...
    col ENUM ('value1', 'value2', 'value3'),
    ...
);
```

在这种语法中，可以有三个以上的枚举值。但是，将枚举值的数量保持在 20 以下是一个很好的做法。

下面来看看下面的例子。

假设我们必须存储优先级为：`low`，`medium` 和 `high` 的票据信息。要将 `priority` 列分配给 `ENUM` 类型，请使用以下`CREATE TABLE`语句：

```
USE testdb;

CREATE TABLE tickets (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    priority ENUM('Low', 'Medium', 'High') NOT NULL
);
```

`priority` 列只接受三个 `Low` , `Medium` , `High` 值。在后台，MySQL 将每个枚举成员映射到数字索引。在这种情况下，`Low` , `Medium` 和 `High` 分别映射到 `1` , `2` 和 `3` (注意：与数组不同，这不是从 `0` 开始的)。

插入 MySQL ENUM 值

要将数据插入到 `ENUM` 列中，可以使用预定义列表中的枚举值。例如，以下语句在 `tickets` 表中插入一个新行。

```
INSERT INTO tickets(title, priority)
VALUES('Scan virus for computer A', 'High');
```

除了枚举值之外，还可以使用枚举成员的数字索引将数据插入 `ENUM` 列。例如，以下语句插入优先级为 `Low` 的新机票数据：

```
INSERT INTO tickets(title, priority)
VALUES('Upgrade Windows OS for all computers', 1);
```

在这个例子中，我们使用的值为：`1`，而不是使用 `Low` 枚举值，因为 `Low` 被映射到 `1`，这是可以接受的。

我们再向 `ticketa` 表添加一些行数据：

```
INSERT INTO tickets(title, priority)
VALUES('Install Google Chrome for Mr. John', 'Medium'),
      ('Create a new user for the new employee David', 'High');
```

因为我们将优先级定义为 `NOT NULL` 列，当插入新行而不指定优先级列的值时，MySQL 将使用第一个枚举成员作为默认值。

参见下列语句声明：

```
INSERT INTO tickets(title)
VALUES('Refresh the computer of Ms. Lily');
```

在非严格的SQL模式下，如果在 `ENUM` 列中插入无效值，MySQL将使用空字符串 `''`，插入数字索引为 `0`。如果启用了严格的SQL模式，尝试插入无效的 `ENUM` 值将导致错误。

请注意，如果 `ENUM` 列定义为可空列，则可以接受 `NULL` 值。

过滤MySQL ENUM值

以下语句查询获得所有高优先级机票：

```
SELECT
*
FROM
    tickets
WHERE
    priority = 'High';
```

执行上面查询语句，得到以下结果 -

<code>id</code>	<code>title</code>	<code>priority</code>
1	Scan virus for computer A	High
4	Create a new user for the new employee David	High

2 rows in set

由于枚举成员' High '映射到 `3`，因此以下查询返回相同的结果集：

```
SELECT
*
FROM
    tickets
WHERE
    priority = 3;
```

也可以使用比较运算符来查询，比如 -

```

SELECT
  *
FROM
  tickets
WHERE
  priority >= 2;

```

排序 MySQL ENUM 值

MySQL 根据索引号 [排序](#) ENUM 值。因此，成员的顺序取决于它们在枚举列表中的定义。

以下查询选择门票并按优先级从高到低进行排序：

```

SELECT
  title, priority
FROM
  tickets
ORDER BY priority DESC;

```

执行上面查询语句，得到以下结果 -

title	priority
Scan virus for computer A	High
Create a new user for the new employee David	High
Install Google Chrome for Mr. John	Medium
Upgrade Windows OS for all computers	Low
Refresh the computer of Ms. Lily	Low

5 rows in set

在创建列时，按顺序定义枚举值是一个很好的做法。

MySQL ENUM 的缺点

MySQL `ENUM` 有以下缺点：

- 更改枚举成员需要使用`ALTER TABLE`语句重建整个表，这在资源和时间方面是昂贵的。
- 获取完整的枚举列表很复杂，因为需要访问

```
information_schema
```

数据库：

```
SELECT
    column_type
FROM
    information_schema.COLUMNS
WHERE
    TABLE_NAME = 'tickets'
        AND COLUMN_NAME = 'priority';
```

- 迁移到其他RDBMS可能是一个问题，因为 `ENUM` 不是SQL标准的，并且数据库系统不支持它。
- 向枚举列表添加更多属性是不可能的。假设您要为每个优先级添加服务协议，例如 `High(24h)`，`Medium(1-2天)`，`Low(1周)`，则不可以使用 `ENUM` 类型的。在这种情况下，需要有一个单独的表来存储优先级列表，例如 `priority(id, name, sort_order, description)`，并且通过引用了 `priority` 表的 `id` 字段的 `priority_id` 来替换 `tickets` 表中的 `priority` 字段。
- 与查找表(`priorities`)相比，枚举列表不可重用。例如，如果要创建一个名为 `tasks` 并且要重用优先级列表的新表，则是不可能的。

在本教程中，我们向您介绍了MySQL `ENUM` 数据类型以及如何使用它来定义存储枚举值的列。

本教程将向您介绍MySQL `NOT NULL` 约束，帮助我们来保持数据的一致性。

MySQL NOT NULL 约束简介

`NOT NULL` 约束是一个列约束，仅将列的值强制为非 `NULL` 值。

`NOT NULL` 约束的语法如下：

```
column_name data_type NOT NULL;
```

列可能只包含一个 `NOT NULL` 约束，它指定列不能包含任何 `NULL` 值。

以下`CREATE TABLE`语句创建 `tasks` 表：

```
USE testdb;
CREATE TABLE tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE
);
```

`title` 和 `start_date` 列明确地显示 `NOT NULL` 约束。`id` 列具有`PRIMARY KEY`约束，因此它隐含地包含一个 `NOT NULL` 约束。

`end_date` 列可以具有 `NULL` 值。这是因为当添加新任务记录时，我们可能还不知道任务的结束日期。

在表的每一列中都有 `NOT NULL` 约束是最好的做法，除非你有很好的理由不使用 `NOT NULL` 约束。

通常，`NULL` 值使查询更加复杂。在这种情况下，可以使用 `NOT NULL` 约束并为列提供默认值。请参阅以下示例：

```
use testdb;
CREATE TABLE inventory (
    material_no VARCHAR(18),
    warehouse_no VARCHAR(10),
    quantity DECIMAL(19, 2) NOT NULL DEFAULT 0,
    base_unit VARCHAR(10) NOT NULL,
    PRIMARY KEY (material_no, warehouse_no)
);
```

在此示例中，数量(quantity)列的默认值为 0 ，因为当向 inventory 表添加一行数据时，数量(quantity)列的值应该为 0 ，而不是 NULL 。

向现有列添加**NOT NULL**约束

通常，在创建表时向列添加 NOT NULL 约束。但是，有时希望将 NOT NULL 约束添加到现有表的 NULL 列。在这种情况下，可以使用以下步骤：

- 检查列的当前值。
- 将所有 NULL 值更新为非 NULL 值。
- 添加 NOT NULL 约束

下面来看一个例子。

我们将数据插入到 tasks 表中以更好地演示。

```
INSERT INTO tasks(title, start_date, end_date)
VALUES('Learn MySQL NOT NULL constraint', '2017-02-01', '2017-02-02'),
      ('Check and update NOT NULL constraint to your database',
       '2017-02-01', NULL);
```

现在，假设要强制用户在创建新任务时给出估计的结束日期。为此，您需要将 NOT NULL 约束添加到 tasks 表的 end_date 列上。

首先检查 end_date 列的值，使用 IS NULL 运算符来检查列中的值是否有 NULL 值：

```

SELECT
*
FROM
tasks
WHERE
end_date IS NULL;

```

执行上面的查询语句，得到以下结果 -

id title start_date end_date s
2 Check and update NOT NULL constraint to your database 2017-02-01 NULL

1 row in set

查询返回 `end_date` 值为 `NULL` 的一行记录。

其次，将 `NULL` 值更新为非 `NULL` 值。在这种情况下，我们可以创建一个规则，如果 `end_date` 为 `NULL`，则在开始日期后一周结束日期。

```

UPDATE tasks
SET
    end_date = start_date + 7
WHERE
    end_date IS NULL;

```

我们来查看一下这个修改后的变化：

```

SELECT
*
FROM
tasks;

```

执行上面的查询语句，得到以下结果 -

```
+----+-----+-----+-----+
| id | title | start_date | end_date | s
+----+-----+-----+-----+
| 1  | Learn MySQL NOT NULL constraint | 2017-02-01 | 2017-02-02 |
| 2  | Check and update NOT NULL constraint to your database | 2017-02-01 | 2017-02-08 |
+----+-----+-----+-----+
2 rows in set
```

第三，将 NOT NULL 约束添加到 end_date 列。要执行此操作，请使用以下 `ALTER TABLE` 语句：

```
ALTER TABLE table_name
CHANGE old_column_name new_column_name new_column_definition;
```

在上面例子中，除了具有 NOT NULL 约束的列定义之外，旧的列名称和新的列名称必须相同：

```
ALTER TABLE tasks
CHANGE end_date end_date DATE NOT NULL;
```

使用 `DESCRIBE` 或 `DESC` 语句来验证更改：

```
mysql> DESC tasks;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id   | int(11) | NO | PRI | NULL | auto_increment |
| title | varchar(255) | NO | | NULL |
| start_date | date | NO | | NULL |
| end_date | date | NO | | NULL |
+-----+-----+-----+-----+-----+
4 rows in set
```

如上所示，NOT NULL 约束被添加到 end_date 列。

在本教程中，您已经学习了如何为列定义 NOT NULL 约束，并将 NOT NULL 约束添加到现有列。

在本教程中，您将学习如何使用MySQL主键(*Primary Key*)约束来创建表的主键。

MySQL主键简介

MySQL主键(*Primary Key*)是唯一标识表中每行的列或一组列。当定义表的主键时，必须遵循以下规则：

- 主键必须包含唯一值。如果主键由多个列组成，则这些列中的值的组合必须是唯一的。
- 主键列不能包含 `NULL` 值。这意味着必须使用 `NOT NULL` 属性声明主键列。如果没有指定 `NOT NULL`，MySQL将强制为主键列为 `NOT NULL`。
- 一张表只有一个主键。

因为MySQL使用整数工作更快，所以主键列的数据类型应该是整数类型，例如：`INT`，`BIGINT`。可以选择一个较小的整数类型：`TINYINT`，`SMALLINT`等。但是，应该确保值的范围的主键的整数类型足以存储表可能所具有最大行数。

主键列通常具有自动生成键的唯一序列的 `AUTO_INCREMENT` 属性。下一行的主键值大于前一个行的主键值。

MySQL为表中的主键创建一个名为 `PRIMARY` 的 `PRIMARY` 索引类型。

定义MySQL主键约束

MySQL允许通过在创建或修改表时定义主键约束来创建主键。

使用`CREATE TABLE`语句定义MySQL **PRIMARY KEY**约束

当使用`CREATE TABLE`语句创建表时，MySQL允许创建主键。要为表创建 `PRIMARY KEY` 约束，请在主键列的定义中指定 `PRIMARY KEY`。

以下示例将为 `users` 表的 `user_id` 列上创建主键：

```
USE testdb;

CREATE TABLE users(
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(40),
    password VARCHAR(255),
    email VARCHAR(255)
);
```

还可以在 CREATE TABLE 语句的末尾指定 PRIMARY KEY ，如下所示：

```
USE testdb;

CREATE TABLE roles(
    role_id INT AUTO_INCREMENT,
    role_name VARCHAR(50),
    PRIMARY KEY(role_id)
);
```

如果主键由多个列组成，则必须在 CREATE TABLE 语句的末尾指定它们。在 PRIMARY KEY 关键字之后，将逗号分隔的主键列的列表在括号内。

```
CREATE TABLE userroles(
    user_id INT NOT NULL,
    role_id INT NOT NULL,
    PRIMARY KEY(user_id,role_id),
    FOREIGN KEY(user_id) REFERENCES users(user_id),
    FOREIGN KEY(role_id) REFERENCES roles(role_id)
);
```

除了创建由 user_id 和 role_id 列组成的主键之外，该语句还创建了两个外键约束。

使用ALTER TABLE语句定义MySQL PRIMARY KEY约束

如果表由于某些原因没有主键，可以使用ALTER TABLE语句将具有所有主键的列添加到主键中，如下语句：

```
ALTER TABLE table_name
ADD PRIMARY KEY(primary_key_column);
```

以下示例将 `id` 列添加到主键。首先，创建 `t1` 表但不定义主键。

```
CREATE TABLE t1(
    id int,
    title varchar(255) NOT NULL
);
```

其次，将 `id` 列作为 `t1` 表的主键。

```
ALTER TABLE t1
ADD PRIMARY KEY(id);
```

PRIMARY KEY与UNIQUE KEY对比

`KEY` 是 `INDEX` 的同义词。当要为列创建索引，但不是主键或唯一键时使用 `KEY`。

`UNIQUE` 索引为其值必须是唯一的列创建约束。与 `PRIMARY` 索引不同，MySQL 在 `UNIQUE` 索引中允许有 `NULL` 值。一个表也可以有多个 `UNIQUE` 索引。

例如，`users` 表中的用户的 `email` 和 `username` 必须是唯一的。可以为 `email` 和 `username` 列定义 `UNIQUE` 索引，如下语句所示：

在 `username` 列上添加 `UNIQUE` 索引。

```
ALTER TABLE users
ADD UNIQUE INDEX username_unique (username ASC);
```

在 `email` 列上添加 `UNIQUE` 索引。

```
ALTER TABLE users
ADD UNIQUE INDEX email_unique (email ASC);
```

在本教程中，您已经学习了如何为新表创建主键或为现有表添加主键。

在本教程中，您将了解MySQL外键(*foreign key*)以及如何在MySQL中创建，添加和删除外键约束。

MySQL外键简介

外键表示一个表中的一个字段被另一个表中的一个字段引用。外键对相关表中的数据造成了限制，使MySQL能够保持参照完整性。

下面来看看示例数据库(yiibaidb)中的以下数据库中两个表：customers 和 `orders` 的ER图。

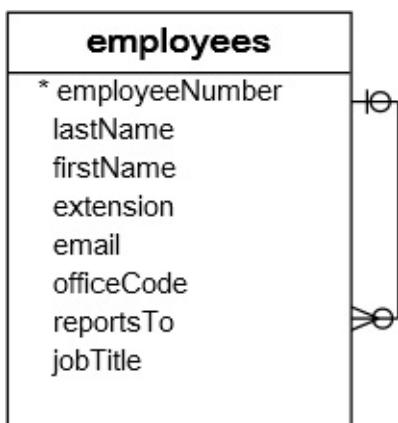
上图中有两张表：customers 和 orders。每个客户有零个或多个订单，每个订单只属于一个客户。customers 表和 orders 表之间的关系是一对多的，它是由 customerNumber 字段指定在 orders 表中建立外键(引用 customers 表的 customerNumber 字段)。orders 表中的 customerNumber 字段与 customers 表中的 customerNumber 主键字段相关。

customers 表称为父表或引用表，orders 表称为子表或引用表。

表可以有多个外键，子表中的每个外键可能引用不同的父表。

子表中的行必须包含父表中存在的值，例如，orders 表中的每个订单记录必须在 customers 表中存在 customerNumber。因此，多个订单可以指同一个客户，因此这种关系称为一个(客户)到许多(订单)或一对多。

有时，子表和父表是一样的。外键返回到表的主键，例如以下 employees 表：



reportTo 列是一个引用 employeeNumber 列的外键，employeeNumber 列是 employees 表的主键，以反映员工之间的报告结构，即每个员工向另一个员工发送的报告和员工可以有零个或多个直接报告。有关如何使用有，请参考[自连接教程](#)，以帮助您根据这种表查询来查询相关数据。

`reportTo` 外键也称为递归或自引用外键。

外键执行引用完整性，可以帮助您自动维护数据的一致性和完整性。例如，不能为不存在的客户创建订单。

此外，可以为 `customerNumber` 外键设置级联删除操作，以便在 `customers` 表中删除客户时，与客户关联的所有订单也将被删除。这样可以节省您使用多个 [DELETE语句](#) 或 [DELETE JOIN语句](#) 的时间和精力。

与删除相同，还可以为 `customerNumber` 外键定义级联更新操作，以执行交叉表更新，而不使用多个[UPDATE语句](#) 或 [UPDATE JOIN语句](#)。

在MySQL中，[InnoDB存储引擎](#) 支持外键，因此您必须创建[InnoDB表](#)才能使用外键约束。

创建外键

MySQL创建外键语法

以下语法说明了如何在[CREATE TABLE语句](#)中的子表中定义外键。

```
CONSTRAINT constraint_name
FOREIGN KEY foreign_key_name (columns)
REFERENCES parent_table(columns)
ON DELETE action
ON UPDATE action
```

下面我们来更详细的查看上面语法：

- `CONSTRAINT` 子句允许您为外键约束定义约束名称。如果省略它，MySQL将自动生成一个名称。
- `FOREIGN KEY` 子句指定子表中引用父表中主键列的列。您可以在 `FOREIGN KEY` 子句后放置一个外键名称，或者让MySQL为您创建一个名称。请注意，MySQL会自动创建一个具有 `foreign_key_name` 名称的索引。
- `REFERENCES` 子句指定父表及其子表中列的引用。在 `FOREIGN KEY` 和 `REFERENCES` 中指定的子表和父表中的列数必须相同。
- `ON DELETE` 子句允许定义当父表中的记录被删除时，子表的记录怎样执行操作。如果省略 `ON DELETE` 子句并删除父表中的记录，则MySQL将拒绝删除子表中相关联的数据。此外，MySQL还提供了一些操作，以便您可以使用其他选

项，例如 `ON DELETE CASCADE`，当删除父表中的记录时，MySQL 可以删除子表中引用父表中记录的记录。如果您不希望删除子表中的相关记录，请改用 `ON DELETE SET NULL` 操作。当父表中的记录被删除时，MySQL 会将子表中的外键列值设置为 `NULL`，条件是子表中的外键列必须接受 `NULL` 值。请注意，如果使用 `ON DELETE NO ACTION` 或 `ON DELETE RESTRICT` 操作，MySQL 将拒绝删除。

- `ON UPDATE` 子句允许指定在父表中的行更新时，子表中的行会怎样执行操作。当父表中的行被更新时，可以省略 `ON UPDATE` 子句让 MySQL 拒绝对子表中的行的任何更新。`ON UPDATE CASCADE` 操作允许您执行交叉表更新，并且当更新父表中的行时，`ON UPDATE SET NULL` 操作会将子表中行中的值重置为 `NULL` 值。`ON UPDATE NO ACTION` 或 `ON UPDATE RESTRICT` 操作拒绝任何更新。

MySQL 创建表外键示例

以下示例创建一个 `dbdemo` 数据库和两个表：`categories` 和 `products`。每个类别都有一个或多个产品，每个产品只属于一个类别。`products` 表中的 `cat_id` 字段被定义为具有 `UPDATE ON CASCADE` 和 `DELETE ON RESTRICT` 操作的外键。

```

CREATE DATABASE IF NOT EXISTS dbdemo;

USE dbdemo;

CREATE TABLE categories(
    cat_id int not null auto_increment primary key,
    cat_name varchar(255) not null,
    cat_description text
) ENGINE=InnoDB;

CREATE TABLE products(
    prd_id int not null auto_increment primary key,
    prd_name varchar(355) not null,
    prd_price decimal,
    cat_id int not null,
    FOREIGN KEY fk_cat(cat_id)
    REFERENCES categories(cat_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT
)ENGINE=InnoDB;

```

添加外键

MySQL添加外键语法

要将外键添加到现有表中，请使用 `ALTER TABLE` 语句与上述外键定义语法：

```

ALTER table_name
ADD CONSTRAINT constraint_name
FOREIGN KEY foreign_key_name(columns)
REFERENCES parent_table(columns)
ON DELETE action
ON UPDATE action;

```

MySQL添加外键示例

现在，我们添加一个名为 `vendors` 的新表，并更改 `products` 表以包含供应商 `ID` 字段：

```
USE dbdemo;

CREATE TABLE vendors(
    vdr_id int not null auto_increment primary key,
    vdr_name varchar(255)
)ENGINE=InnoDB;

ALTER TABLE products
ADD COLUMN vdr_id int not null AFTER cat_id;
```

要在 `products` 表中添加外键，请使用以下语句：

```
ALTER TABLE products
ADD FOREIGN KEY fk_vendor(vdr_id)
REFERENCES vendors(vdr_id)
ON DELETE NO ACTION
ON UPDATE CASCADE;
```

现在，`products` 表有两个外键，一个是引用 `categories` 表，另一个是引用 `vendors` 表。

删除MySQL外键

您还可以使用 `ALTER TABLE` 语句将外键删除，如下语句：

```
ALTER TABLE table_name
DROP FOREIGN KEY constraint_name;
```

在上面的声明中：

- 首先，指定要从中删除外键的表名称。
- 其次，将约束名称放在 `DROP FOREIGN KEY` 子句之后。

请注意，`constraint_name` 是在创建或添加外键到表时指定的约束的名称。如果省略它，MySQL会为您生成约束名称。

要获取生成的表的约束名称，请使用 `SHOW CREATE TABLE` 语句，如下所示：

```
SHOW CREATE TABLE table_name;
```

例如，要查看 `products` 表的外键，请使用以下语句：

```
SHOW CREATE TABLE products;
```

以下是语句的输出：

```
CREATE TABLE products (
    prd_id int(11) NOT NULL AUTO_INCREMENT,
    prd_name varchar(355) NOT NULL,
    prd_price decimal(10,0) DEFAULT NULL,
    cat_id int(11) NOT NULL,
    vdr_id int(11) NOT NULL,
    PRIMARY KEY (prd_id),
    KEY fk_cat (cat_id),
    KEY fk_vendor(vdr_id),

    CONSTRAINT products_ibfk_2
    FOREIGN KEY (vdr_id)
    REFERENCES vendors (vdr_id)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,

    CONSTRAINT products_ibfk_1
    FOREIGN KEY (cat_id)
    REFERENCES categories (cat_id)
    ON UPDATE CASCADE
) ENGINE=InnoDB;
```

`products` 表有两个外键约束：`products_ibfk_1` 和 `products_ibfk_2`。

可以使用以下语句删除 `products` 表的外键：

```
ALTER TABLE products
DROP FOREIGN KEY products_ibfk_1;

ALTER TABLE products
DROP FOREIGN KEY products_ibfk_2;
```

MySQL禁用外键检查

有时，因为某种原因需要禁用外键检查(例如[将CSV文件中的数据导入表中](#))非常有用。如果不禁用外键检查，则必须以正确的顺序加载数据，即必须首先将数据加载到父表中，然后再将数据加载导入到子表中，这可能是乏味的。但是，如果禁用外键检查，则可以按任何顺序加载导入数据。

除非禁用外键检查，否则不能删除由外键约束引用的表。删除表时，还会删除为表定义的任何约束。

要禁用外键检查，请使用以下语句：

```
SET foreign_key_checks = 0;
```

当然，可以使用以下语句启用它：

```
SET foreign_key_checks = 1;
```

在本教程中，我们已经介绍了很多有关MySQL外键的内容。还向您介绍了一些非常方便的语句，允许您在MySQL中有效地管理外键。

在本教程中，您将了解MySQL UNIQUE约束，以强制一列或一个组合列的值的唯一性。

MySQL UNIQUE约束简介

有时，希望在列中强制执行唯一性值，例如供应商表中供应商的电话必须是唯一的，或者供应商名称和地址的组合不得重复，简单一点理解就是：供应商的名称可以相同，但是不能在同一个地址。

要执行此规则，需要使用 `UNIQUE` 约束。

`UNIQUE` 约束是列约束或表约束，它定义了限制列或一组列中的值为唯一的规则。

要将 `UNIQUE` 约束添加到列，请使用以下语法：

```
CREATE TABLE table_1(
    column_name_1 data_type UNIQUE,
);
```

或者可以将 `UNIQUE` 约束定义为表约束，如下所示：

```
CREATE TABLE table_1(
    ...
    column_name_1 data_type,
    ...
    UNIQUE(column_name_1)
);
```

如果在 `column_name_1` 列中插入或更新导致重复值的值，MySQL将发出错误消息并拒绝更改。

如果要跨列强制执行唯一值，则必须将 `UNIQUE` 约束定义为表约束，并将每列用逗号分隔：

```
CREATE TABLE table_1(
    ...
    column_name_1 data_type,
    column_name_2 data type,
    ...
    UNIQUE(column_name_1, column_name_2)
);
```

MySQL将使用 `column_name_1` 和 `column_name_2` 列中的值的组合来评估唯一性。

如果要为 `UNIQUE` 约束分配一个指定的名称，可以使用 `CONSTRAINT` 子句，如下所示：

```
CREATE TABLE table_1(
    ...
    column_name_1 data_type,
    column_name_2 data type,
    ...
    CONSTRAINT constraint_name UNIQUE(column_name_1, column_name_2)
);
```

MySQL UNIQUE约束示例

以下语句创建了一个名为 `suppliers` 的新表，具有两个唯一约束条件：

```
USE testdb;
CREATE TABLE IF NOT EXISTS suppliers (
    supplier_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(16) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    CONSTRAINT uc_name_address UNIQUE (name , address)
);
```

第一个 `UNIQUE` 约束应用于 `phone` 列。这意味着每个供应商必须具有不同的电话号码。换句话说，没有哪两个供应商的电话号码是相同的。第二个 `UNIQUE` 约束的名称为 `uc_name_address`，它强制 `name` 和 `address` 列中值的唯一性。供应商可以拥有相同的名称或地址，但名称和地址不能同时相同。

我们在 `suppliers` 表中插入一些行来测试 `UNIQUE` 约束。

以下语句将一行插入到 `suppliers` 表中。

```
INSERT INTO suppliers(name, phone, address)
VALUES('ABC Inc', '13800138000', '4000 North 1st Street, San Jose,
, CA, USA');
```

尝试插入不同的供应商数据信息，但是使用的电话号码是一个在 `suppliers` 表中已经存在电话号码。

```
INSERT INTO suppliers(name, phone, address)
VALUES('XYZ Corporation', '13800138000', '4001 North 1st Street,
San Jose, CA, USA');
```

MySQL发出错误：

```
Error Code: 1062. Duplicate entry '13800138000' for key 'phone'
```

我们将电话号码更改为其他号码，然后再次执行插入语句。

```
INSERT INTO suppliers(name, phone, address)
VALUES('XYZ Corporation', '13800138111', '400 North 1st Street, S
an Jose, CA, USA');
```

上面插入语句可以成功执行。

现在执行以下 `INSERT` 语句来插入一行，其中已经存在的名称和地址列中的值。

```
INSERT INTO suppliers(name, phone, address)
VALUES('XYZ Corporation', '13800138222', '400 North 1st Street, S
an Jose, CA, USA');
```

MySQL发出以下错误信息 -

```
1062 - Duplicate entry 'XYZ Corporation-400 North 1st Street, San Jose, CA, USA' for key 'uc_name_address'
```

因为违反了 `UNIQUE` 约束 `uc_name_address`，所以插入失败。

管理MySQL UNIQUE约束

当您向表中添加唯一约束时，MySQL将为数据库创建一个相应的 `BTREE` 索引。以下 `SHOW INDEX` 语句显示在 `suppliers` 表上创建的所有索引。

```
SHOW INDEX FROM testdb.suppliers;
```

结果如下所示 -

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
▶	suppliers	0	PRIMARY	1	supplier_id	A	0	NULL	NULL		BTREE
①	suppliers	0	phone	1	phone	A	0	NULL	NULL		BTREE
②	suppliers	0	uc_name_address	1	name	A	0	NULL	NULL		BTREE
	suppliers	0	uc_name_address	2	address	A	0	NULL	NULL		BTREE

如上图所见，有两个 `BTREE` 索引对应于创建的两个 `UNIQUE` 约束。

要删除 `UNIQUE` 约束，可以使用 `DROP INDEX` 或 `ALTER TABLE` 语句，语法如下所示：

```
DROP INDEX index_name ON table_name;
```

或者

```
ALTER TABLE table_name
DROP INDEX index_name;
```

例如，要删除 `suppliers` 表上的 `uc_name_address` 约束，需要以下语句：

```
DROP INDEX uc_name_address ON suppliers;
```

再次执行 `SHOW INDEX` 语句来验证 `uc_name_unique` 约束是否已经被删除。

```
SHOW INDEX FROM testdb.suppliers;
```

执行上面查询语句，得到以下结果

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
▶	suppliers	0	PRIMARY	1	supplier_id	A	0	NULL	NULL		BTREE
	suppliers	0	phone	1	phone	A	0	NULL	NULL		BTREE

如果要将 `UNIQUE` 约束添加到已存在的表中，该怎么办？很简单，使用 `ALTER TABLE` 语句，语法如下所示：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name UNIQUE (column_list);
```

例如，要将名称为 `uc_name_address` 的 `UNIQUE` 约束添加到 `suppliers` 表，请使用以下语句：

```
ALTER TABLE suppliers
ADD CONSTRAINT uc_name_address UNIQUE (name,address);
```

请注意，为了使语句成功执行，当然表的 `name` 和 `address` 列中的值的组合必须是唯一的。

在本教程中，您已经学习了如何使用MySQL `UNIQUE` 约束来强制表中列或一组列中的值的唯一性。

在本教程中，您将学习如何使用带有 `check` 选项的触发器或视图来模拟MySQL `CHECK` 约束。

注意：要更好学习和理解本教程，您需要对触发器，视图和存储过程有很好的了解。

SQL CHECK约束简介

标准SQL提供的检查(`CHECK`)约束指定某列中的值必须满足布尔表达式。例如，您可以添加一个 `CHECK` 约束来强制成本(`cost`)列为正值，如下所示：

```
USE testdb;
CREATE TABLE IF NOT EXISTS parts (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10 , 2 ) NOT NULL CHECK(cost > 0),
    price DECIMAL (10,2) NOT NULL
);
```

SQL允许您将一个或多个CHECK约束应用于一列或跨多个列。例如，为了确保价格(`price`)列总是大于或等于成本(`cost`)列，可使用 `CHECK` 约束如下：

```
USE testdb;
CREATE TABLE IF NOT EXISTS parts (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10 , 2 ) NOT NULL CHECK (cost > 0),
    price DECIMAL(10 , 2 ) NOT NULL CHECK (price > 0),
    CHECK (price >= cost)
);
```

当 `CHECK` 约束设置完成，每当插入或更新导致布尔表达式的值计算为 `false` 时，则视为违反检查约束，并且数据库系统拒绝插入或更改数据。

不幸的是，MySQL不支持 `CHECK` 约束。实际上，MySQL在`CREATE TABLE`语句中接受 `CHECK` 子句，但是它会以静默方式忽略它。

MySQL 使用触发器CHECK约束

在MySQL中模拟 CHECK 约束的第一种方法是使用两个触发器： BEFORE INSERT 和 BEFORE UPDATE 。

首先，为了演示目的，我们先创建一个 parts 表，如下语句 -

```
USE testdb;
CREATE TABLE IF NOT EXISTS parts (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10 , 2 ) NOT NULL,
    price DECIMAL(10,2) NOT NULL
);
```

其次，创建一个存储过程来检查 cost 和 price 列中的值。

```

DELIMITER $$

CREATE PROCEDURE `check_parts` (IN cost DECIMAL(10,2), IN price DECIMAL(10,2))
BEGIN
    IF cost < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'check constraint on parts.cost failed';
    END IF;

    IF price < 0 THEN
        SIGNAL SQLSTATE '45001'
        SET MESSAGE_TEXT = 'check constraint on parts.price failed';
    END IF;

    IF price < cost THEN
        SIGNAL SQLSTATE '45002'
        SET MESSAGE_TEXT = 'check constraint on parts.price & parts.cost failed';
    END IF;

END$$

DELIMITER ;

```

第三，创建 BEFORE INSERT 和 BEFORE UPDATE 触发器。在触发器中，调用 check_parts() 存储过程。

```
-- before insert
DELIMITER $$

CREATE TRIGGER `parts_before_insert` BEFORE INSERT ON `parts`
FOR EACH ROW
BEGIN
    CALL check_parts(new.cost, new.price);
END$$
DELIMITER ;

-- before update
DELIMITER $$

CREATE TRIGGER `parts_before_update` BEFORE UPDATE ON `parts`
FOR EACH ROW
BEGIN
    CALL check_parts(new.cost, new.price);
END$$
DELIMITER ;
```

第四，插入满足以下所有条件的新行：

- cost > 0
- price > 0
- price >= cost

执行以下插入语句 -

```
INSERT INTO parts(part_no, description, cost, price)
VALUES('A-001', 'Cooler', 100, 120);
```

INSERT 语句调用 BEFORE INSERT 触发器并接受值。

演示-1 以下 INSERT 语句执行将会失败，因为它违反了条件： cost> 0 。

```
INSERT INTO parts(part_no, description, cost, price)
VALUES('A-002', 'Heater', -100, 120);
```

执行上面插入语句，得到以下错误提示信息 -

Error Code: 1644. check constraint on parts.cost failed

演示-2 以下 INSERT 语句执行将会失败，因为它违反了条件： price > 0 。

```
INSERT INTO parts(part_no, description, cost, price)
VALUES('A-002', 'Heater', 100, -120);
```

执行上面插入语句，得到以下错误提示信息 -

Error Code: 1644. check constraint on parts.price failed

演示-3 以下 INSERT 语句执行将会失败，因为它违反了条件： price > cost 。

```
INSERT INTO parts(part_no, description, cost, price)
VALUES('A-003', 'wiper', 120, 100);
```

执行上面插入语句，得到以下错误提示信息 -

1644 - check constraint on parts.price & parts.cost failed

现在，让我们来看看在 parts 表中的数据。

```
SELECT * FROM parts;
```

执行上面查询语句，得到以下结果 -

part_no	description	cost	price
A 001	Cooler	100	120

1 row in set

我们试图更新 cost 列的值，使其低于价格(price)列：

```
UPDATE parts
SET price = 10
WHERE part_no = 'A-001';
```

执行上面更新语句，得到以下错误提示信息 -

```
Error Code: 1644. check constraint on parts.price & parts.cost failed
```

上面更新语句被拒绝执行了。

如上示例中所示，我们通过使用两个触发器：`BEFORE INSERT` 和 `BEFORE UPDATE`，来模拟MySQL中的 `CHECK` 约束。

MySQL CHECK约束使用可更新视图与check选项

这个方法是使用基于表的 `check` 选项来创建一个视图。在视图的 `SELECT` 语句中，我们仅选择满足 `CHECK` 条件的有效行。对视图的任何插入或更新都将被拒绝，这样使新的行记录不会出现在视图中。

首先，删除 `parts` 表以删除所有相关的触发器，并创建一个新的表，与 `parts` 表具有相同的结构，但使用了不同的名称：`parts_data`：

```
DROP TABLE IF EXISTS parts;

CREATE TABLE IF NOT EXISTS parts_data (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10, 2) NOT NULL,
    price DECIMAL(10, 2) NOT NULL
);
```

其次，根据 `parts_data` 表创建名为 `parts` 的视图。通过这样做，我们可以保持使用 `parts` 表的应用程序的代码保持不变。此外，旧零件表的所有权限保持不变。

```

CREATE VIEW vparts AS
SELECT
    part_no, description, cost, price
FROM
    parts_data
WHERE
    cost > 0 AND price > 0 AND price >= cost
WITH CHECK OPTION;

```

第三，通过 `parts` 视图向 `parts_data` 表中插入一个新行：

```

INSERT INTO vparts(part_no, description, cost, price)
VALUES('A-001', 'Cooler', 100, 120);

```

上面的新行可被接受，因为新行有效，可以出现在视图中。

但是，以下语句失败，因为新行不会出现在视图中。

```

INSERT INTO vparts(part_no, description, cost, price)
VALUES('A-002', 'Heater', -100, 120);

```

执行上面语句，MySQL会发出以下错误 -

```
Error Code: 1369. CHECK OPTION failed 'testdb.parts_checked'
```

在本教程中，我们向您介绍了标准SQL `CHECK` 约束和两种在MySQL中模拟 `CHECK` 约束的方法用法。

在本教程中，您将了解MySQL中的字符集。在本教程之后，您将了解如何获取MySQL中的所有字符集，如何在字符集之间转换字符串以及如何为客户端连接配置正确的字符集。

MySQL字符集简介

MySQL字符集是一组在字符串中合法的字符。例如，我们有一个从 `a` 到 `z` 的字母。要为每个字母分配一个数字，例如 `a = 1`，`b = 2` 等。字母 `a` 是一个符号，数字 `1` 与字母 `a` 相关联就是一种编码。从 `a` 到 `z` 的所有字母和它们相应的编码的组合是一个字符集。

每个字符集具有一个或多个排序规则，其定义用于比较字符集中的字符的一组规则。查看[MySQL排序规则教程](#)，了解MySQL排序规则。

MySQL支持各种字符集，允许您几乎可将每个字符存储在字符串中。要获取MySQL数据库服务器中的所有可用字符集，请使用 `SHOW CHARACTER SET` 语句如下：

<code>mysql> SHOW CHARACTER SET;</code>		
<code>Charset</code>	<code>Description</code>	<code>Default collation</code>
	<code>Maxlen</code>	
<code>big5</code>	<code>Big5 Traditional Chinese</code>	<code>big5_chinese_ci</code>
	<code>2</code>	
<code>dec8</code>	<code>DEC West European</code>	<code>dec8_swedish_ci</code>
	<code>1</code>	
<code>cp850</code>	<code>DOS West European</code>	<code>cp850_general_ci</code>
	<code>1</code>	
<code>hp8</code>	<code>HP West European</code>	<code>hp8_english_ci</code>
	<code>1</code>	
<code>koi8r</code>	<code>KOI8_R Relcom Russian</code>	<code>koi8r_general_ci</code>
	<code>1</code>	
<code>latin1</code>	<code>cp1252 West European</code>	<code>latin1_swedish_ci</code>
	<code>1</code>	
<code>latin2</code>	<code>ISO 8859-2 Central European</code>	<code>latin2_general_ci</code>
	<code>1</code>	

swe7	7bit Swedish	swe7_swedish_ci
1		
ascii	US ASCII	ascii_general_ci
1		
ujis	EUC JP Japanese	ujis_japanese_ci
3		
sjis	Shift JIS Japanese	sjis_japanese_ci
2		
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci
1		
tis620	TIS620 Thai	tis620_thai_ci
1		
euckr	EUC KR Korean	euckr_korean_ci
2		
koi8u	KOI8-U Ukrainian	koi8u_general_ci
1		
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci
2		
greek	ISO 8859-7 Greek	greek_general_ci
1		
cp1250	Windows Central European	cp1250_general_ci
1		
gbk	GBK Simplified Chinese	gbk_chinese_ci
2		
latin5	ISO 8859-9 Turkish	latin5_turkish_ci
1		
armSCII8	ARMSCII-8 Armenian	armSCII8_general_ci
1		
utf8	UTF 8 Unicode	utf8_general_ci
3		
ucs2	UCS-2 Unicode	ucs2_general_ci
2		
cp866	DOS Russian	cp866_general_ci
1		
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci
1		
macce	Mac Central European	macce_general_ci
1		
macroman	Mac West European	macroman_general_ci
1		
cp852	DOS Central European	cp852_general_ci
1		

latin7	ISO 8859-13 Baltic	latin7_general_ci
i	1	
utf8mb4	UTF-8 Unicode	utf8mb4_general_c
i	4	
cp1251	Windows Cyrillic	cp1251_general_ci
i	1	
utf16	UTF-16 Unicode	utf16_general_ci
i	4	
utf16le	UTF-16LE Unicode	utf16le_general_c
i	4	
cp1256	Windows Arabic	cp1256_general_ci
i	1	
cp1257	Windows Baltic	cp1257_general_ci
i	1	
utf32	UTF-32 Unicode	utf32_general_ci
i	4	
binary	Binary pseudo charset	binary
i	1	
geostd8	GEOSTD8 Georgian	geostd8_general_c
i	1	
cp932	SJIS for Windows Japanese	cp932_japanese_ci
i	2	
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_c
i	3	
gb18030	China National Standard GB18030	gb18030_chinese_ci
i	4	
-----+-----+-----+		
-----+-----+-----+		
41 rows in set		

MySQL中的默认字符集是 latin1 。如果要在单列中存储多种语言的字符，可以使用 Unicode 字符集，即 utf8 或 ucs2 。

Maxlen 列中的值指定字符集中的字符持有的字节数。一些字符集包含单字节字符，例如： latin1 ， latin2 ， cp850 等，而其他字符集包含多字节字符。

MySQL提供了 LENGTH 函数来获取字节的长度，以字节为单位， CHAR_LENGTH 函数用于获取字符串的长度。如果字符串包含多字节字符，则 LENGTH 函数的结果大于 CHAR_LENGTH() 函数的结果。请参阅以下示例：

```
SET @str = CONVERT('我的MySQL' USING ucs2);

SELECT LENGTH(@str), CHAR_LENGTH(@str);
```

执行上面查询，得到以下结果 -

LENGTH(@str)	CHAR_LENGTH(@str)
14	7

1 row in set

`CONVERT` 函数将字符串转换为指定的字符集。在这个例子中，它将 MySQL 字符集字符串的字符集转换为 `ucs2`。因为 `ucs2` 字符集包含 2 个字节的字符，因此 `@str` 字符串的长度(以字节为单位)大于其字符长度。

请注意，某些字符集包含多字节字符，但其字符串可能只包含单字节字符，例如 `utf8`，如以下语句所示：

```
SET @str = CONVERT('MySQL Character Set' USING utf8);
SELECT LENGTH(@str), CHAR_LENGTH(@str);
```

执行上面查询语句，得到以下结果 -

LENGTH(@str)	CHAR_LENGTH(@str)
19	19

1 row in set

但是，如果 `utf8` 字符串包含特殊字符，例如 `ü` 在 `pingüino` 字符串中；其字节长度不同，请参见以下示例：

```
SET @str = CONVERT('pingüino' USING utf8);
SELECT LENGTH(@str), CHAR_LENGTH(@str);
```

执行上面查询语句，得到以下结果 -

LENGTH(@str)	CHAR_LENGTH(@str)
9	8

1 row in set

一个使用中文的示例 -

```
SET @str = CONVERT('不要问我有多长' USING utf8);
SELECT LENGTH(@str), CHAR_LENGTH(@str);
```

执行上面查询语句，得到以下结果 -

LENGTH(@str)	CHAR_LENGTH(@str)
21	7

1 row in set

在使用 `utf8` 字符集编码时，中文字占 3 个长度。

转换不同的字符集

MySQL提供了两个函数，允许您在不同字符集之间转换字符串：`CONVERT` 和 `CAST`。在上面的例子中，我们多次使用了 `CONVERT` 函数。

`CONVERT` 函数的语法如下：

```
CONVERT(expression USING character_set_name)
```

CAST函数类似于 CONVERT 函数。它将字符串转换为不同的字符集：

```
CAST(string AS character_type CHARACTER SET character_set_name)
```

看一下使用 CAST 函数的以下示例：

```
SELECT CAST(_latin1'MySQL character set' AS CHAR CHARACTER SET utf8);
```

执行上面查询语句，得到以下结果 -

CAST(_latin1'MySQL character set' AS CHAR CHARACTER SET utf8)
+-----+
MySQL character set
+-----+
1 row in set

设置客户端连接的字符集

当应用程序与MySQL数据库服务器交换数据时，默认字符集为 latin1 。但是，如果数据库在 utf8 字符集中存储Unicode字符串，则使用应用程序中的 latin1 字符集将是不够的。因此，当连接到MySQL数据库服务器时，应用程序需要指定正确的字符集。

要配置客户端连接的字符集，可以执行以下方式之一：

- 客户端连接到MySQL数据库服务器后发出 SET NAME 语句。例如，要设置 Unicode字符集 utf8 ，请使用以下语句：

```
SET NAMES 'utf8';
```

- 如果应用程序支持 `--default-character-set` 选项，则可以使用它来设置字符集。例如，mysql客户端工具支持 `--default-character-set`，您可以在配置文件中进行如下设置：

```
[mysql]
default-character-set=utf8
```

- 一些MySQL连接器允许您设置字符集，例如，如果您使用PHP PDO，则可以按如下方式设置数据源名称中的字符集：

```
$dsn = "mysql:host=$host;dbname=$db;charset=utf8";
```

无论使用哪种方式，请确保应用程序使用的字符集与存储在MySQL数据库服务器中的字符集匹配。

在本教程中，您已经了解了MySQL字符集，如何在字符集之间转换字符串以及如何为客户端连接配置正确的字符集。

在本教程中，您将了解MySQL校对规则以及如何设置MySQL服务器，数据库，表和列的字符集和校对规则。

MySQL校对规则简介

MySQL校对规则是用于比较特定字符集中的字符的一组规则。MySQL中的每个字符集可以有多个校对规则，并且至少具有一个默认校对规则。两个字符集不能具有相同的归类。

MySQL提供了 `SHOW CHARACTER SET` 语句，查看字符集的默认校对规则，如下所示：

Charset	Description	Default collation
Maxlen		
big5	Big5 Traditional Chinese	big5_chinese_ci
2		
dec8	DEC West European	dec8_swedish_ci
1		
cp850	DOS West European	cp850_general_ci
1		
hp8	HP West European	hp8_english_ci
1		
koi8r	KOI8-R Relcom Russian	koi8r_general_ci
1		
latin1	cp1252 West European	latin1_swedish_ci
1		
latin2	ISO 8859-2 Central European	latin2_general_ci
1		
swe7	7bit Swedish	swe7_swedish_ci
1		
ascii	US ASCII	ascii_general_ci
1		
ujis	EUC-JP Japanese	ujis_japanese_ci
3		
sjis	Shift-JIS Japanese	sjis_japanese_ci
2		

hebrew	ISO 8859-8 Hebrew	hebrew_general_ci
	1	
tis620	TIS620 Thai	tis620_thai_ci
	1	
euckr	EUC KR Korean	euckr_korean_ci
	2	
koi8u	KOI8-U Ukrainian	koi8u_general_ci
	1	
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci
	2	
greek	ISO 8859-7 Greek	greek_general_ci
	1	
cp1250	Windows Central European	cp1250_general_ci
	1	
gbk	GBK Simplified Chinese	gbk_chinese_ci
	2	
latin5	ISO 8859-9 Turkish	latin5_turkish_ci
	1	
armSCII8	ARMSCII-8 Armenian	armSCII8_general_ci
	1	
utf8	UTF-8 Unicode	utf8_general_ci
	3	
ucs2	UCS-2 Unicode	ucs2_general_ci
	2	
cp866	DOS Russian	cp866_general_ci
	1	
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci
i	1	
macce	Mac Central European	macce_general_ci
	1	
macroman	Mac West European	macroman_general_ci
ci	1	
cp852	DOS Central European	cp852_general_ci
	1	
latin7	ISO 8859-13 Baltic	latin7_general_ci
	1	
utf8mb4	UTF-8 Unicode	utf8mb4_general_ci
i	4	
cp1251	Windows Cyrillic	cp1251_general_ci
	1	
utf16	UTF-16 Unicode	utf16_general_ci
	4	

	utf16le	UTF 16LE Unicode	utf16le_general_c
i		4	
	cp1256	Windows Arabic	cp1256_general_ci
		1	
	cp1257	Windows Baltic	cp1257_general_ci
		1	
	utf32	UTF 32 Unicode	utf32_general_ci
		4	
	binary	Binary pseudo charset	binary
		1	
	geostd8	GEOSTD8 Georgian	geostd8_general_c
i		1	
	cp932	SJIS for Windows Japanese	cp932_japanese_ci
		2	
	eucjpms	UJIS for Windows Japanese	eucjpms_japanese_
ci		3	
	gb18030	China National Standard GB18030	gb18030_chinese_c
i		4	
-----+-----+-----+-----			
-----+-----+-----+-----			
41 rows in set			

默认校对规则列的值指定字符集的默认校对规则。

按照惯例，字符集的校对规则以字符集名称开头，以 `_ci` (不区分大小写) `_cs` (区分大小写)或 `_bin` (二进制文件)结尾。

要获取给定字符集的所有校对规则，请使用 `SHOW COLLATION` 语句如下：

```
SHOW COLLATION LIKE 'character_set_name%';
```

例如，要获取 `latin1` 字符集的所有校对规则，请使用以下语句：

```
SHOW COLLATION LIKE 'latin1%';
```

执行上面语句，得到用于 `latin1` 字符集的MySQL校对规则，如下结果 -

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+
--+
| latin1_german1_ci | latin1 | 5 | Yes | Yes | 1 |
| latin1_swedish_ci | latin1 | 8 | Yes | Yes | 1 |
| latin1_danish_ci | latin1 | 15 | Yes | Yes | 1 |
| latin1_german2_ci | latin1 | 31 | Yes | Yes | 2 |
| latin1_bin | latin1 | 47 | Yes | Yes | 1 |
| latin1_general_ci | latin1 | 48 | Yes | Yes | 1 |
| latin1_general_cs | latin1 | 49 | Yes | Yes | 1 |
| latin1_spanish_ci | latin1 | 94 | Yes | Yes | 1 |
+-----+-----+-----+-----+
--+
8 rows in set
```

如上所述，每个字符集都具有默认校对规则，例

如 `latin1_swedish_ci` 是 `latin1` 字符集的默认校对规则。

设置字符集和校对规则

MySQL允许您在四个级别指定字符集和校对规则：服务器，数据库，表和列。

在服务器级别设置字符集和校对规则

注意MySQL使用 `latin1` 作为默认字符集，因此，其默认校对规则为 `latin1_swedish_ci`。您可以在服务器启动时更改这些设置。

如果在服务器启动时仅指定一个字符集，MySQL将使用字符集的默认校对规则。如果您明确指定了一个字符集和校对规则，MySQL将使用数据库服务器中的字符集和校对规则来创建的所有数据库。

以下语句通过命令行启动并设置服务器使用 `utf8` 字符集和 `utf8_unicode_ci` 校对规则：

```
$ mysql - --character-set-server=utf8 --collation-server=utf8_unicode_ci
```

在数据库级别设置字符集和校对规则

创建数据库时，如果不指定其字符集和校对规则，MySQL将使用数据库的服务器的默认字符集和校对规则。

可以使用[CREATE DATABASE](#)或[ALTER DATABASE](#)语句来覆盖数据库级的默认设置，如下所示：

```
CREATE DATABASE database_name  
CHARACTER SET character_set_name  
COLLATE collation_name  
  
-- 修改校对规则  
ALTER DATABASE database_name  
CHARACTER SET character_set_name  
COLLATE collation_name
```

MySQL在数据库级使用数据库中创建的所有表的字符集和校对规则。

在表级别设置字符集和校对规则

数据库可能包含与默认数据库的字符集和校对规则不同的字符集和校对规则的表。

当您通过使用 `CREATE TABLE` 语句创建表或使用 `ALTER TABLE` 语句更改表的结构时，可以指定表的默认字符集和校对规则。

```
CREATE TABLE table_name(  
)  
CHARACTER SET character_set_name  
COLLATE collation_name
```

或者 -

```
ALTER TABLE table_name(  
)  
CHARACTER SET character_set_name  
COLLATE collation_name
```

在列级别设置字符集和校对规则

CHAR , VARCHAR 或 TEXT 类型的列可以使用与表的默认字符集和校对规则不同的，自己指定的字符集和校对规则。

可以按照 CREATE TABLE 或 ALTER TABLE 语句的列定义中的列指定字符集和校对规则，如下所示：

```
column_name [CHAR | VARCHAR | TEXT] (length)  
CHARACTER SET character_set_name  
COLLATE collation_name
```

以下是设置字符集和校对规则的规则：

- 如果显式指定字符集和校对规则，则使用字符集和校对规则。
- 如果指定一个字符集并忽略校对规则，则使用字符集的默认校对规则。
- 如果指定没有字符集的校对规则，则使用与校对规则相关联的字符集。
- 如果省略字符集和校对规则，则使用默认字符集和校对规则。

我们来看一些设置字符集和校对规则的例子。

设置字符集和校对规则的示例

首先，我们使用 utf8 作为字符集创建一个新数据库，将 utf8_unicode_ci 作为默认校对规则：

```
CREATE DATABASE mydbdemo
CHARACTER SET utf8
COLLATE utf8_unicode_ci;
```

因为明确指定 `mydbdemo` 数据库的字符集和校对规则，所以 `mydbdemo` 数据库不会在服务器级别采用默认字符集和校对规则。

其次，我们在 `mydbdemo` 数据库中创建一个名为 `t1` 的新表，但不指定字符集和校对规则：

```
USE mydbdemo;
CREATE TABLE t1(
    c1 char(25)
);
```

如上所示，我们并没有为 `t1` 表指定字符集和校对规则；MySQL 将检查数据库级别以确定 `t1` 表的字符集和校对规则。在这种情况下，`t1` 表将使用 `utf8` 作为默认字符集，`utf8_unicode_ci` 作为默认校对规则。

第三，对于 `t1` 表，我们将其字符集更改为 `latin1`，并将其校对规则改为 `latin1_german1_ci`：

```
ALTER TABLE t1
CHARACTER SET latin1
COLLATE latin1_german1_ci;
```

`t1` 表中的 `c1` 列使用 `latin1` 作为字符集，`latin1_german1_ci` 作为校对规则。

第四，将 `c1` 列的字符集更改为 `latin1`：

```
ALTER TABLE t2
MODIFY c1 VARCHAR(25)
CHARACTER SET latin1;
```

现在，`c1` 列使用 `latin1` 字符集，但是它的校对规则呢？是否从表的校对规则继承了 `latin1_german1_ci` 校对规则？不是的，因为 `latin1` 字符集的默认校对规则是 `latin1_swedish_ci`，所以 `c1` 列具有 `latin1_swedish_ci` 校对规则。

在本教程中，您已经了解了MySQL校对规则以及如何为MySQL服务器，数据库，表和列指定字符集和校对规则。

- 将CSV文件导入MySQL表 - 演示如何使用 `LOAD DATA INFILE` 语句将CSV文件导入MySQL表。
- MySQL导出表到CSV - 学习如何将MySQL表导出为CSV文件格式的各种技术。

本教程您学习如何使用 `LOAD DATA INFILE` 语句将CSV文件导入到MySQL表中。
`LOAD DATA INFILE` 语句允许您从文本文件读取数据，并将文件的数据快速导入数据库的表中。

在导入文件操作之前，需要准备以下内容：

- 将要导入文件的数据对应的数据库表。
- 准备好一个CSV文件，其数据与表的列数和每列中的数据类型相匹配。
- 连接到MySQL数据库服务器的帐户具有 `FILE` 和 `INSERT` 权限。

假设我们有一个名为 `discounts` 表，具有以下结构：

discounts	
id	INT(11)
title	VARCHAR(255)
expired_date	DATE
amount	DECIMAL(10,2)
Indexes	

接下来，使用`CREATE TABLE`语句创建 `discounts` 表，如下所示：

```
use testdb;
CREATE TABLE discounts (
    id INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    expired_date DATE NOT NULL,
    amount DECIMAL(10 , 2 ) NULL,
    PRIMARY KEY (id)
);
```

以下是 `discounts.csv` 文件的内容，第一行作为列标题和其他三行则为数据。

```
id,title,expired date,amout
1,"Spring Break 2018",20180401,20
2,"Back to Scholl 2017",20170901,29
3,"Summer 2018",20180820,100
```

以下语句将数据从 `F:/worksp/mysql/discounts.csv` 文件导入到 `discounts` 表。

16.1 导入 CSV 文件

```
LOAD DATA INFILE 'F:/worksp/mysql/discounts.csv'  
INTO TABLE discounts  
FIELDS TERMINATED BY ','  
ENCLOSED BY '\"'  
LINES TERMINATED BY '\n'  
IGNORE 1 ROWS;
```

文件的字段由 `FIELD TERMINATED BY ','` 指示的逗号终止，并由 `ENCLOSED BY '\"'` 指定的双引号括起来。

因为文件第一行包含列标题，列标题不需要导入到表中，因此通过指定 `IGNORE 1 ROWS` 选项来忽略第一行。

现在，我们可以查看 `discounts` 表中的数据，查看是否成功导入了数据。

```
SELECT * FROM discounts;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT * FROM discounts;  
+----+-----+-----+-----+  
| id | title | expired_date | amount |  
+----+-----+-----+-----+  
| 1 | Spring Break 2018 | 2018-04-01 | 20 |  
| 2 | Back to Scholl 2017 | 2017-09-01 | 29 |  
| 3 | Summer 2018 | 2018-08-20 | 100 |  
+----+-----+-----+-----+  
3 rows in set
```

导入时转换数据

有时，数据格式与表中的目标列不匹配。在简单的情况下，可以使用 `LOAD DATA INFILE` 语句中的 `SET` 子句进行转换。

假设有一个 `discount_2.csv` 文件中，它存储的过期日期列是 `mm/dd/yyyy` 格式。其内容如下所示 -

```

id,title,expired_date,amount
4,"Item-4","01/04/2018",200
5,"Item-5","01/09/2017",290
6,"Item-6","12/08/2018",122

```

将数据导入 `discounts` 表时，必须使用 `str_to_date()` 函数将其转换为 MySQL 日期格式，如下所示：

```

LOAD DATA INFILE 'F:/worksp/mysql/discounts_2.csv'
INTO TABLE discounts
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id,title,@expired_date,amount)
SET expired_date = STR_TO_DATE(@expired_date, '%m/%d/%Y');

```

现在查询表中的数据，得到以下结果 -

```

mysql> SELECT * FROM discounts;
+----+-----+-----+-----+
| id | title          | expired_date | amount |
+----+-----+-----+-----+
| 1  | Spring Break 2018 | 2018-04-01   | 20     |
| 2  | Back to Scholl 2017 | 2017-09-01   | 29     |
| 3  | Summer 2018      | 2018-08-20   | 100    |
| 4  | Item-4           | 2018-01-04   | 200    |
| 5  | Item-5           | 2017-01-09   | 290    |
| 6  | Item-6           | 2018-12-08   | 122    |
+----+-----+-----+-----+
6 rows in set

```

将文件从客户端导入远程 MySQL 数据库服务器

可以使用 `LOAD DATA INFILE` 语句将数据从客户端(本地计算机)导入远程MySQL数据库服务器。

当您在 `LOAD DATA INFILE` 中使用 `LOCAL` 选项时，客户端程序会读取客户端上的文件并将其发送到MySQL服务器。该文件将被上传到数据库服务器操作系统的临时文件夹，例如Windows上的 `C:\windows\temp` 或Linux上为 `/tmp` 目录。此文件夹不可由MySQL配置或确定。

我们来看看下面的例子：

```
LOAD DATA LOCAL INFILE 'c:/tmp/discounts.csv'  
INTO TABLE discounts  
FIELDS TERMINATED BY ','  
ENCLOSED BY '\"'  
LINES TERMINATED BY '\n'  
IGNORE 1 ROWS;
```

唯一的区别是语句中多了个 `LOCAL` 选项。如果加载一个大的CSV文件，将会看到使用 `LOCAL` 选项来加载该文件将会稍微慢些，因为需要时间将文件传输到数据库服务器。

使用 `LOCAL` 选项时，连接到MySQL服务器的帐户不需要具有 `FILE` 权限来导入文件。

使用 `LOAD DATA LOCAL` 将文件从客户端导入到远程数据库服务器时，有一些安全问题应该要注意，以避免潜在的安全风险。

在本教程中，您将学习如何将MySQL表导出到CSV文件的各种方法/技术。

CSV代表逗号分隔值。您经常使用CSV文件格式在 *Microsoft Excel*，*Open Office*，*Google Docs* 等应用程序之间交换数据。

以CSV文件格式从MySQL数据库中获取数据将非常有用，因为您可以按照所需的方式分析和格式化数据。

MySQL提供了一种将查询结果导出到位于数据库服务器中的CSV文件的简单方法。

在导出数据之前，必须确保：

- MySQL服务器的进程对包含目标CSV文件的目标文件夹具有写访问权限。
- 要导出的目标CSV文件不能存在。

以下查询从 `orders` 表中查询选择已取消的订单：

```
SELECT
    orderNumber, status, orderDate, requiredDate, comments
FROM
    orders
WHERE
    status = 'Cancelled';
```

要将此结果集导出为CSV文件，请按如下方式向上述查询添加一些子句：

```
SELECT
    orderNumber, status, orderDate, requiredDate, comments
FROM
    orders
WHERE
    status = 'Cancelled'
INTO OUTFILE 'F:/worksp/mysql/cancelled_orders.csv'
FIELDS ENCLOSED BY ''
TERMINATED BY ';'
ESCAPED BY ''
LINES TERMINATED BY '\r\n';
```

该语句在 `F:/worksp/mysql/` 目录下创建一个包含结果集，名称为 `cancelled_orders.csv` 的CSV文件。

CSV文件包含结果集中的行集合。每行由一个回车序列和由 `LINES TERMINATED BY '\r\n'` 子句指定的换行字符终止。文件中的每行包含表的结果集的每一行记录。

每个值由 `FIELDS ENCLOSED BY '"'` 子句指示的双引号括起来。这样可以防止可能包含逗号(,)的值被解释为字段分隔符。当用双引号括住这些值时，该值中的逗号不会被识别为字段分隔符。

将数据导出到文件名包含时间戳的CSV文件

我们经常需要将数据导出到CSV文件中，该文件的名称包含创建文件的时间戳。为此，您需要使用[MySQL准备语句](#)。

以下命令将整个 `orders` 表导出为将时间戳作为文件名的一部分的CSV文件。

```
SET @TS = DATE_FORMAT(NOW(), '_%Y%m%d_%H%i%s');

SET @FOLDER = 'F:/worksp/mysql/';
SET @PREFIX = 'orders';
SET @EXT = '.csv';

SET @CMD = CONCAT("SELECT * FROM orders INTO OUTFILE ''",@FOLDER,
@PREFIX,@TS,@EXT,
"' FIELDS ENCLOSED BY '\"' TERMINATED BY ';' ESCAPED BY '\"''"
,
"' LINES TERMINATED BY '\r\n';");

PREPARE statement FROM @CMD;

EXECUTE statement;
```

下面，让我们来详细讲解上面的命令。

- 首先，构造了一个具有当前时间戳的查询作为文件名的一部分。
- 其次，使用 `PREPARE` 语句 `FROM` 命令准备执行语句。
- 第三，使用 `EXECUTE` 命令执行语句。

可以通过[事件](#)包装命令，并根据需要定期安排事件的运行。

使用列标题导出数据

如果CSV文件包含第一行作为列标题，那么该文件更容易理解，这是非常方便的。

要添加列标题，需要使用**UNION**语句如下：

```
(SELECT 'Order Number', 'Order Date', 'Status')
UNION
(SELECT orderNumber, orderDate, status
FROM orders
INTO OUTFILE 'F:/worksp/mysql/orders_union_title.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ';' ESCAPED BY '"'
LINES TERMINATED BY '\r\n');
```

如查询所示，需要包括每列的列标题。

处理**NULL**值

如果结果集中的值包含**NULL**值，则目标文件将使用“**N/A**”来代替数据中的**NULL**值。要解决此问题，您需要将**NULL**值替换为另一个值，例如不适用(**N/A**)，方法是使用**IFNULL**函数，如下：

```
SELECT
    orderNumber, orderDate, IFNULL(shippedDate, 'N/A')
FROM
    orders INTO OUTFILE 'F:/worksp/mysql/orders_null2na.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ';'
ESCAPED BY '"' LINES
TERMINATED BY '\r\n';
```

我们用**N/A**字符串替换了**shippingDate**列中的**NULL**值。CSV文件将显示**N/A**而不是**NULL**值。

第二章 技巧

在本节中，我们为您提供高级的MySQL技术和技巧，以帮助您有效地解决MySQL中一些棘手的难题。

MySQL CTE简介

- 本教程将向您展示如何使用MySQL CTE或公用表表达式功能以更可读的方式构建复杂查询。参阅：<http://www.yiibai.com/mysql/cte.html>

MySQL递归CTE的最终指南

- 在本教程中，您将了解MySQL递归CTE以及如何使用它来遍历MySQL数据库中的分层数据。参阅：<http://www.yiibai.com/mysql/recursive-cte.html>

使用邻接列表模型管理MySQL中的分层数据

- 在本教程中，您将学习如何使用邻接列表模型来管理MySQL中的层次结构数据。参阅：<http://www.yiibai.com/mysql/adjacency-list-tree.html>

MySQL行计数：如何在MySQL中获取表中的行数？

- 本教程将向您展示在MySQL数据库中获取MySQL行计数的各种方法。参阅：<http://www.yiibai.com/mysql/row-count.html>

MySQL比较两个表

- 本教程将向您展示如何比较两个表，以查找MySQL中不匹配的记录。参阅：<http://www.yiibai.com/mysql/compare-two-tables-to-find-unmatched-records-mysql.html>

如何在MySQL中找到重复的值

- 本教程将逐步介绍如何使用纯SQL语句在MySQL中的一列或多列中找到重复值。参阅：<http://www.yiibai.com/mysql/find-duplicate-values.html>

如何删除MySQL中的重复行

- 在本教程中，您将学习如何使用 `DELETE JOIN` 语句或即时表删除MySQL中的重复行。参阅：<http://www.yiibai.com/mysql/delete-duplicate-rows.html>

MySQL UUID : UUID 与 INT 作为主键的比较

- 本教程将向您介绍MySQL UUID，演示如何将其用作表的主键(PK)，并讨论将其用作PK的优缺点。参阅：<http://www.yiibai.com/mysql/uuid.html>

MySQL 复制表示例

- 本教程将向您展示如何使用MySQL中的 CREATE TABLE LIKE 和 SELECT 语句将表在同一数据库中，或者将表从一个数据库复制到另一个数据库中。参阅：<http://www.yiibai.com/mysql/copy-table-data.html>

如何复制 MySQL 数据库

- 本教程将向您展示如何在同一服务器，或在另外一个服务器上复制MySQL数据库。参阅：<http://www.yiibai.com/mysql/copy-database.html>

MySQL 变量

- 在本教程中，您将学习如何在SQL语句中使用MySQL用户定义的变量。参阅：<http://www.yiibai.com/mysql/variables.html>

如何使用 MySQL 生成的列

- 在本教程中，将学习如何使用MySQL生成的列来存储从表达式或其他列计算的数据。参阅：<http://www.yiibai.com/mysql/generated-columns.html>

如何比较 MySQL 中相同表中的连续行

- 在本文章中，我们将通过在MySQL中使用 INNER JOIN 向您展示如何比较或计算连续行之间的差异。参阅：<http://www.yiibai.com/mysql/compare-calculate-difference-successive-rows.html>

如何更改 MySQL 存储引擎

- 在本教程中，您将了解表如何使用哪个存储引擎，以及如何将表的存储引擎更改为其他引擎。参阅：<http://www.yiibai.com/mysql/change-storage-engine.html>

MySQL REGEXP：基于正则表达式的搜索

- 在本教程中，您将了解正则表达式以及如何使用MySQL REGEXP 运算符使用许多正则表达式示例查询数据。参阅：<http://www.yiibai.com/mysql/regular-expression-regexp.html>

MySQL row_number函数以及如何模拟它

- 在本教程中，我们将向您展示一个模拟MySQL中 `row_number` 函数的使用技巧。参阅：http://www.yiibai.com/mysql/row_number.html

MySQL选择随机记录

- 显示从数据库表中选择随机记录的各种技术。参阅：<http://www.yiibai.com/mysql/select-random-records-database-table.html>

如何选择MySQL中第n个最高纪录

- 在本教程中，您将学习如何使用各种技术在数据库表中第 `n` 个最高记录。参阅：<http://www.yiibai.com/mysql/select-nth-highest-record-database-table-using-mysql.html>

MySQL重置自动增量值

- 本教程将向您展示如何重置MySQL中 `AUTO_INCREMENT` 列的自动增量值。参阅：<http://www.yiibai.com/mysql/reset-auto-increment.html>

MariaDB与MySQL比较区别

- 下表说明了MariaDB和MySQL之间的主要区别：MySQL MariaDBDeveloper Oracle Corporation MariaDB Corporation AB(MariaDB Enterprise)，MariaDB Foundation(社区MariaDB Server)协议... 参阅：<http://www.yiibai.com/mysql/mariadb-vs-mysql.html>

MySQL间隔

- 本教程将逐步介绍如何使用MySQL间隔进行日期和时间算术与许多实际示例。参阅：<http://www.yiibai.com/mysql/interval.html>

MySQL NULL：初学者指南

- 在本教程中，您将学习如何使用MySQL `NULL` 值。此外，还将学习一些有用的函数来有效地处理 `NULL` 值。参阅：<http://www.yiibai.com/mysql/null.html>

如何获取MySQL今天的日期

- 本教程将向您展示如何通过使用 `CURDATE()` 或 `NOW()` 函数来获取MySQL今天的日期。另外，还将学习如何创建 `today()` 存储函数。参阅：<http://www.yiibai.com/mysql/today.html>

如何将**NULL**值映射到其他有意义的值

- 您将学习如何将 **NULL** 值映射到其他值以获得更好的数据表示。参考：<http://www.yiibai.com/mysql/avoid-displaying-null-values-by-mapping-to-other-values.html>

MySQL注释深度

- 本教程将向您展示如何使用MySQL注释来记录SQL语句或代码块。参考：<http://www.yiibai.com/mysql/comment.html>

在本教程中，您将学习如何使用MySQL CTE或公用表表达式以更可读的方式构建复杂查询。

自MySQL 8.0版以来简要介绍了公共表表达式或叫**CTE**的功能，因此需要您在计算机上安装MySQL 8.0，以便在本教程中练习本语句。

1. 什么是公用表表达式或**CTE**？

公用表表达式是一个命名的临时结果集，仅在单个SQL语句(例如 **SELECT**，**INSERT**，**UPDATE**或**DELETE**)的执行范围内存在。

与派生表类似，**CTE**不作为对象存储，仅在查询执行期间持续。与派生表不同，**CTE**可以是自引用(递归**CTE**)，也可以在同一查询中多次引用。此外，与派生表相比，**CTE**提供了更好的可读性和性能。

2. MySQL CTE语法

CTE的结构包括名称，可选列列表和定义**CTE**的查询。定义**CTE**后，可以像 **SELECT**，**INSERT**，**UPDATE**，**DELETE** 或 **CREATE VIEW** 语句中的视图一样使用它。

以下说明了**CTE**的基本语法：

```
WITH cte_name (column_list) AS (
    query
)
SELECT * FROM cte_name;
```

请注意，查询中的列数必须与 **column_list** 中的列数相同。如果省略 **column_list**，**CTE**将使用定义**CTE**的查询的列列表。

3. 简单的MySQL CTE示例

以下示例说明如何使用**CTE**查询[示例数据库\(yiibaidb\)](#)中的 **customers** 表中的数据。请注意，此示例仅用于演示目的，以便您更容易地了解**CTE**概念。

```

WITH customers_in_usa AS (
    SELECT
        customerName, state
    FROM
        customers
    WHERE
        country = 'USA'
) SELECT
    customerName
FROM
    customers_in_usa
WHERE
    state = 'CA'
ORDER BY customerName;

```

注意：上面语句只能在 MySQL8.0 以上版本才支持。

执行上面查询语句，得到以下结果(部分)

	customerName
▶	Boards & Toys Co.
	Collectable Mini Designs Co.
	Corporate Gift Ideas Co.
	Men 'R' US Retailers, Ltd.
	Mini Gifts Distributors Ltd.
	Mini Wheels Co.
	Signal Collectibles Ltd.
	Technics Stores Inc.
	The Sharp Gifts Warehouse
	Toys4GrownUps.com
	West Coast Collectables Co.

在此示例中，**CTE**的名称为 `customers_in_usa`，定义**CTE**的查询返回两列：`customerName` 和 `state`。因此，`customers_in_usa CTE`返回位于美国的所有客户。

在定义美国**CTE**的客户之后，我们可在 `SELECT` 语句中引用它，例如，仅查询选择位于 *California* 的客户。

参见另外一个例子：

```

WITH topsales2013 AS (
    SELECT
        salesRepEmployeeNumber employeeNumber,
        SUM(quantityOrdered * priceEach) sales
    FROM
        orders
        INNER JOIN
        orderdetails USING (orderNumber)
        INNER JOIN
        customers USING (customerNumber)
    WHERE
        YEAR(shippedDate) = 2013
        AND status = 'Shipped'
    GROUP BY salesRepEmployeeNumber
    ORDER BY sales DESC
    LIMIT 5
)
SELECT
    employeeNumber, firstName, lastName, sales
FROM
    employees
    JOIN
    topsales2013 USING (employeeNumber);

```

执行上面查询后，得到以下结果 -

	employeeNumber	firstName	lastName	sales
▶	1165	Leslie	Jennings	413219.85
	1370	Gerard	Hernandez	295246.44
	1401	Pamela	Castillo	289982.88
	1621	Mami	Nishi	267249.40
	1501	Larry	Bott	261536.95

在这个例子中，CTE中返回了在2013年前五名的销售代表。之后，我们引用了 topsales2013 CTE来获取有关销售代表的其他信息，包括名字和姓氏。

4. 更高级的MySQL CTE示例

请参阅以下示例：

```

WITH salesrep AS (
    SELECT
        employeeNumber,
        CONCAT(firstName, ' ', lastName) AS salesrepName
    FROM
        employees
    WHERE
        jobTitle = 'Sales Rep'
),
customer_salesrep AS (
    SELECT
        customerName, salesrepName
    FROM
        customers
        INNER JOIN
        salesrep ON employeeNumber = salesrepEmployeeNumber
)
SELECT
    *
FROM
    customer_salesrep
ORDER BY customerName;

```

执行上面查询语句，得到以下结果 -

	customerName	salesrepName
▶	Alpha Cognac	Gerard Hernandez
	American Souvenirs Inc	Foon Yue Tseng
	Amica Models & Co.	Pamela Castillo
	Anna's Decorations, Ltd	Andy Fixter
	Atelier graphique	Gerard Hernandez
	Australian Collectables, Ltd	Andy Fixter
	Australian Collectors, Co.	Andy Fixter
	Australian Gift Network, Co	Andy Fixter
	Auto Associés & Cie.	Gerard Hernandez
	Auto Canal+ Petit	Loui Bondur
	Auto-Moto Classics Inc.	Steve Patterson
	AV Stores, Co.	Larry Bott

在这个例子中，在同一查询中有两个CTE。第一个CTE(salesrep)获得职位是销售代表的员工。第二个CTE(customer_salesrep)使用 INNER JOIN 子句与第一个CTE连接来获取每个销售代表负责的客户。

在使用第二个**CTE**之后，使用带有**ORDER BY**子句的简单 **SELECT** 语句来查询来自该**CTE**的数据。

5. WITH子句用法

有一些上下文可以使用 **WITH** 子句来创建公用表表达式(**CTE**)：

首先，在 **SELECT**，**UPDATE** 和 **DELETE** 语句的开头可以使用 **WITH** 子句：

```
WITH ... SELECT ...
WITH ... UPDATE ...
WITH ... DELETE ...
```

第二，可以在子查询或派生表子查询的开头使用 **WITH** 子句：

```
SELECT ... WHERE id IN (WITH ... SELECT ...);

SELECT * FROM (WITH ... SELECT ...) AS derived_table;
```

第三，可以在 **SELECT** 语句之前立即使用 **WITH** 子句，包括 **SELECT** 子句：

```
CREATE TABLE ... WITH ... SELECT ...
CREATE VIEW ... WITH ... SELECT ...
INSERT ... WITH ... SELECT ...
REPLACE ... WITH ... SELECT ...
DECLARE CURSOR ... WITH ... SELECT ...
EXPLAIN ... WITH ... SELECT ...
```

在本教程中，您已经学会了如何使用MySQL 公共表表达式(**CTE**)来构造复杂的查询语句。

在本教程中，您将了解MySQL递归CTE(公共表表达式)以及如何使用它来遍历分层数据。

自MySQL 8.0版以来简要介绍了公共表表达式或叫CTE的功能，因此需要您在计算机上安装MySQL 8.0，以便在本教程中练习本语句。

1. MySQL递归CTE简介

递归公用表表达式(CTE)是一个具有引用CTE名称本身的子查询的CTE。以下说明递归CTE的语法 -

```
WITH RECURSIVE cte_name AS (
    initial_query -- anchor member
    UNION ALL
    recursive_query -- recursive member that references to the C
    TE name
)
SELECT * FROM cte_name;
```

递归CTE由三个主要部分组成：

- 形成CTE结构的基本结果集的初始查询(*initial_query*)，初始查询部分被称为锚成员。
- 递归查询部分是引用CTE名称的查询，因此称为递归成员。递归成员由一个UNION ALL或 UNION DISTINCT运算符与锚成员相连。
- 终止条件是当递归成员没有返回任何行时，确保递归停止。

递归CTE的执行顺序如下：

1. 首先，将成员分为两个：锚点和递归成员。
2. 接下来，执行锚成员形成基本结果集(R_0)，并使用该基本结果集进行下一次迭代。
3. 然后，将 R_i 结果集作为输入执行递归成员，并将 R_{i+1} 作为输出。
4. 之后，重复第三步，直到递归成员返回一个空结果集，换句话说，满足终止条件。
5. 最后，使用 UNION ALL 运算符将结果集从 R_0 到 R_n 组合。

2. 递归成员限制

递归成员不能包含以下结构：

- 聚合函数，如 `MAX`, `MIN`, `SUM`, `AVG`, `COUNT` 等
- `GROUP BY` 子句
- `ORDER BY` 子句
- `LIMIT` 子句
- `DISTINCT`

请注意，上述约束不适用于锚定成员。另外，只有在使用 `UNION` 运算符时，要禁止 `DISTINCT` 才适用。如果使用 `UNION DISTINCT` 运算符，则允许使用 `DISTINCT`。

另外，递归成员只能在其子句中引用 `CTE` 名称，而不是引用任何 `子查询`。

3. 简单的 MySQL 递归 CTE 示例

请参阅以下简单的递归 `CTE` 示例：

```
WITH RECURSIVE cte_count (n)
AS (
    SELECT 1
    UNION ALL
    SELECT n + 1
    FROM cte_count
    WHERE n < 3
)
SELECT n
FROM cte_count;
```

在此示例中，以下查询：

```
SELECT 1
```

是作为基本结果集返回 `1` 的锚成员。

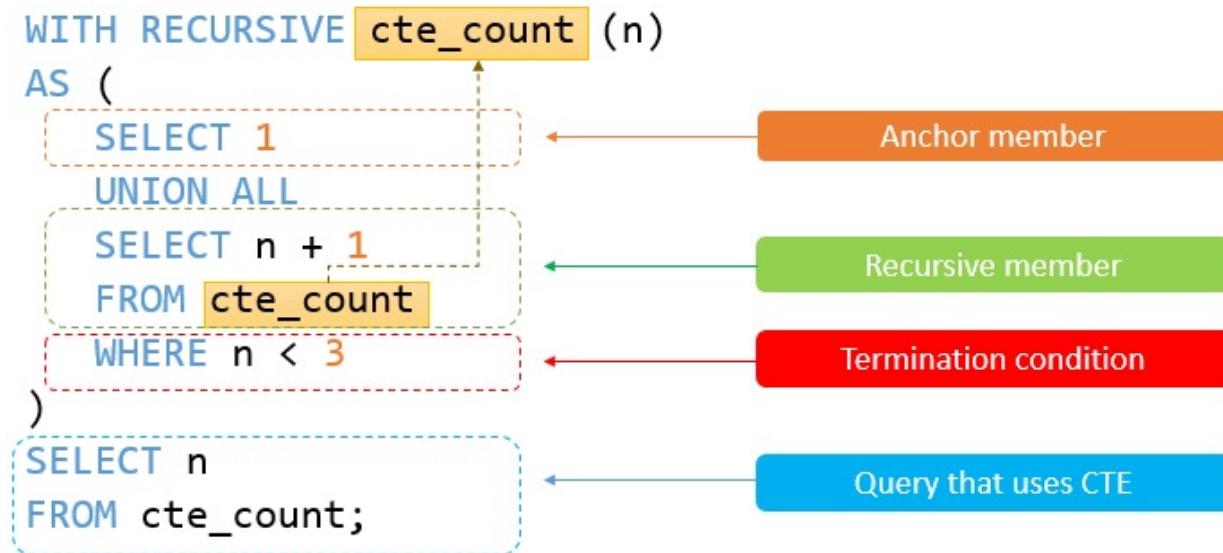
以下查询 -

```
SELECT n + 1
FROM cte_count
WHERE n < 3
```

是递归成员，因为它引用了 `cte_count` 的 **CTE** 名称。

递归成员中的表达式 `<3` 是终止条件。当 `n` 等于 `3`，递归成员将返回一个空集合，将停止递归。

下图显示了上述 **CTE** 的元素：



递归 **CTE** 返回以下输出：

	n
▶	1
	2
	3

递归 **CTE** 的执行步骤如下：

- 首先，分离锚和递归成员。
- 接下来，锚定成员形成初始行(`SELECT 1`)，因此第一次迭代在 `n = 1` 时产生 `1 + 1 = 2`。
- 然后，第二次迭代对第一次迭代的输出(`2`)进行操作，并且在 `n = 2` 时产生 `2 + 1 = 3`。
- 之后，在第三次操作(`n = 3`)之前，满足终止条件(`n < 3`)，因此查询停止。
- 最后，使用 `UNION ALL` 运算符组合所有结果集 `1, 2` 和 `3`。

4. 使用MySQL递归CTE遍历分层数据

我们将使用示例数据库(yiibaidb)中的 employees 表进行演示。

```
mysql> desc employees;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| employeeNumber | int(11) | NO | PRI | NULL |
| lastName | varchar(50) | NO | | NULL |
| firstName | varchar(50) | NO | | NULL |
| extension | varchar(10) | NO | | NULL |
| email | varchar(100) | NO | | NULL |
| officeCode | varchar(10) | NO | MUL | NULL |
| reportsTo | int(11) | YES | MUL | NULL |
| jobTitle | varchar(50) | NO | | NULL |
+-----+-----+-----+-----+-----+
8 rows in set
```

employees 表具有引用 employeeNumber 字段的 reportsTo 字段。

reportsTo 列存储经理的 ID 。总经理不会向公司的组织结构中的任何人报告，因此 reportsTo 列中的值为NULL。

您可以应用递归CTE以自顶向下的方式查询整个组织结构，如下所示：

```

WITH RECURSIVE employee_paths AS
( SELECT employeeNumber,
         reportsTo managerNumber,
         officeCode,
         1 lvl
   FROM employees
 WHERE reportsTo IS NULL
 UNION ALL
   SELECT e.employeeNumber,
          e.reportsTo,
          e.officeCode,
          lvl+1
    FROM employees e
   INNER JOIN employee_paths ep ON ep.employeeNumber = e.reportsTo )
SELECT employeeNumber,
       managerNumber,
       lvl,
       city
  FROM employee_paths ep
 INNER JOIN offices o USING (officeCode)
 ORDER BY lvl, city;

```

让我们将查询分解成更小的部分，使其更容易理解。首先，使用以下查询形成锚成员：

```

SELECT
  employeeNumber, reportsTo managerNumber, officeCode
FROM
  employees
WHERE
  reportsTo IS NULL

```

此查询(锚成员)返回 `reportTo` 为 `NULL` 的总经理。

其次，通过引用 **CTE** 名称来执行递归成员，在这个示例中为 `employee_paths`：

```

SELECT
    e.employeeNumber, e.reportsTo, e.officeCode
FROM
    employees e
        INNER JOIN
    employee_paths ep ON ep.employeeNumber = e.reportsTo

```

此查询(递归成员)返回经理的所有直接上级，直到没有更多的直接上级。如果递归成员不返回直接上级，则递归停止。

第三，使用 `employee_paths` 的查询将 `CTE` 返回的结果集与 `offices` 表结合起来，以得到最终结果集合。

以下是查询的输出：

	employeeNumber	managerNumber	lvl	city
▶	1002	NULL	1	San Francisco
	1076	1002	2	San Francisco
	1056	1002	2	San Francisco
	1102	1056	3	Paris
	1143	1056	3	San Francisco
	1088	1056	3	Sydney
	1621	1056	3	Tokyo
	1188	1143	4	Boston
	1216	1143	4	Boston
	1504	1102	4	London
	1501	1102	4	London
	1286	1143	4	NYC
	1323	1143	4	NYC
	1401	1102	4	Paris
	1702	1102	4	Paris
	1337	1102	4	Paris
	1370	1102	4	Paris
	1166	1143	4	San Francisco

在本教程中，您已经了解了MySQL递归 `CTE` 以及如何使用它来遍历分层数据。

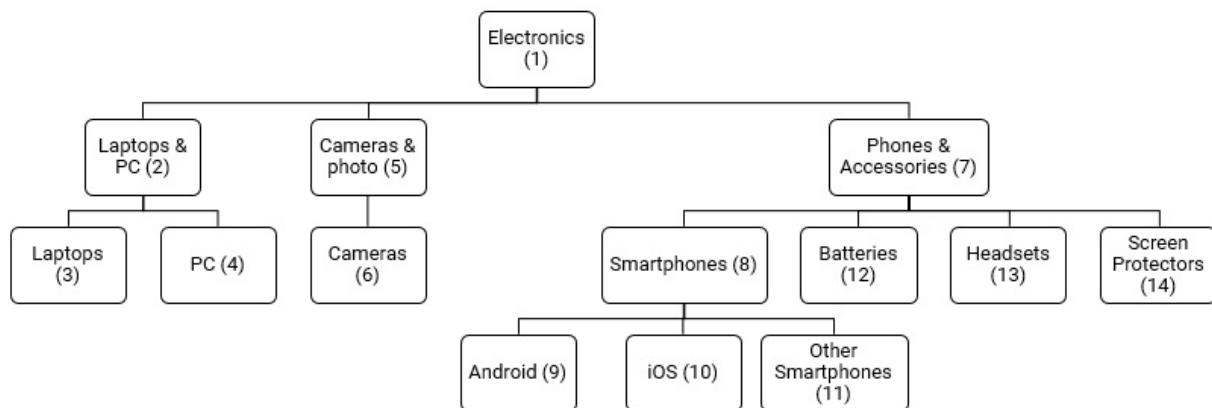
在本教程中，您将学习如何使用邻接列表模型来管理MySQL中的分层数据。

邻接列表模型介绍

分层数据无处不在。它可以是博客类别(栏目)，产品层次结构或组织结构。

有很多方法来管理MySQL中的层次数据，邻接列表模型可能是最简单的解决方案。由于其简单性，邻接列表模型是开发人员和数据库管理员非常受欢迎的选择。

在邻接列表模型中，每个节点都有一个指向其父节点的指针。顶级节点没有父节点。请参阅以下类别的电子产品：



在使用邻接列表模型之前，应该熟悉一些术语：

- 电子设备(`Electronics`)是顶级节点或根节点。
- 笔记本电脑，相机和照片，手机和配件(`Laptops`, `Cameras & photo`, `Phones & Accessories`)节点是 `Electronics` 节点的子节点。反之亦然 `Electronics` 节点是 `Laptops`, `Cameras & photo`, `Phones & Accessories` 节点的父节点。
- 叶子节点是没有子节点的节点，例如 `Laptops` , `PC` , `Android` , `iOS` 等，而非叶节点是至少有一个子节点的节点。
- 一个节点的子孙节点被称为后代节点。一个节点的父节点，祖父节点等也被称为祖先节点。

要对此类树进行建模，我们可以创建一个名为 `category` 的表，其中包含三个列： `id` , `title` 和 `parent_id` ，如下所示：

```
CREATE TABLE category (
    id int(10) unsigned NOT NULL AUTO_INCREMENT,
    title varchar(255) NOT NULL,
    parent_id int(10) unsigned DEFAULT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (parent_id) REFERENCES category (id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

表中的每一行都是由 `id` 列标识的树中的一个节点。`parent_id` 列是 `category` 表本身的外键。它像一个指向 `id` 列的指针。

插入数据

树的根节点没有父节点，因此 `parent_id` 设置为 `NULL`。其他节点必须只有一个父节点。

要插入根节点数据，请将 `parent_id` 设置为 `NULL`，如下所示：

```
INSERT INTO category(title,parent_id)
VALUES('Electronics',NULL);
```

要插入非根节点，只需要将其 `parent_id` 设置为其父节点的 ID 值。例如，`Laptop & PC` 和 `Cameras & Photos`，以及 `Phone & Accessories` 节点的 `parent_id` 设置为 `1`，参考以下语句：

```
INSERT INTO category(title, parent_id)
VALUES('Laptops & PC', 1);

INSERT INTO category(title, parent_id)
VALUES('Laptops', 2);
INSERT INTO category(title, parent_id)
VALUES('PC', 2);

INSERT INTO category(title, parent_id)
VALUES('Cameras & photo', 1);
INSERT INTO category(title, parent_id)
VALUES('Camera', 5);

INSERT INTO category(title, parent_id)
VALUES('Phones & Accessories', 1);
INSERT INTO category(title, parent_id)
VALUES('Smartphones', 7);

INSERT INTO category(title, parent_id)
VALUES('Android', 8);
INSERT INTO category(title, parent_id)
VALUES('iOS', 8);
INSERT INTO category(title, parent_id)
VALUES('Other Smartphones', 8);

INSERT INTO category(title, parent_id)
VALUES('Batteries', 7);
INSERT INTO category(title, parent_id)
VALUES('Headsets', 7);
INSERT INTO category(title, parent_id)
VALUES('Screen Protectors', 7);
```

查找根节点

根节点是没有父节点的节点。换句话说，它的 `parent_id` 为 `NULL`：

```
SELECT
    id, title
FROM
    category
WHERE
    parent_id IS NULL;
```

查找节点的直接子节点

以下查询获取根节点的直接子节点，参考以下查询语句 -

```
SELECT
    id, title
FROM
    category
WHERE
    parent_id = 1;
```

查找叶节点

叶节点是没有子节点的节点。

```
SELECT
    c1.id, c1.title
FROM
    category c1
    LEFT JOIN
    category c2 ON c2.parent_id = c1.id
WHERE
    c2.id IS NULL;
```

查询整个树

以下[递归公用表表达式\(CTE\)](#)检索整个类别树。请注意，自从MySQL 8.0起，CTE功能已经可用了。

```

WITH RECURSIVE category_path (id, title, path) AS
(
    SELECT id, title, title as path
    FROM category
    WHERE parent_id IS NULL
UNION ALL
    SELECT c.id, c.title, CONCAT(cp.path, ' > ', c.title)
    FROM category_path AS cp JOIN category AS c
    ON cp.id = c.parent_id
)
SELECT * FROM category_path
ORDER BY path;

```

查询子树

以下查询获取ID为 7 的 Phone&Accessories 的子树。

```

WITH RECURSIVE category_path (id, title, path) AS
(
    SELECT id, title, title as path
    FROM category
    WHERE parent_id = 7
UNION ALL
    SELECT c.id, c.title, CONCAT(cp.path, ' > ', c.title)
    FROM category_path AS cp JOIN category AS c
    ON cp.id = c.parent_id
)
SELECT * FROM category_path
ORDER BY path;

```

得到以下结果 -

	id	title	path
▶	12	Batteries	Batteries
	13	Headsets	Headsets
	14	Screen Protectors	Screen Protectors
	8	Smartphones	Smartphones
	9	Android	Smartphones > Android
	10	iOS	Smartphones > iOS
	11	Other Smartphones	Smartphones > Other Smartphones

查询单个路径

要查询从下到上的单一路径，例如从 `iOS` 到 `Electronics`，请使用以下语句：

```
WITH RECURSIVE category_path (id, title, parent_id) AS
(
    SELECT id, title, parent_id
    FROM category
    WHERE id = 10 -- child node
UNION ALL
    SELECT c.id, c.title, c.parent_id
    FROM category_path AS cp JOIN category AS c
    ON cp.parent_id = c.id
)
SELECT * FROM category_path;
```

计算每个节点的级别

假设根节点的级别为 `0`，下面的每个节点都有一个等于其父节点的级别加 `1` 的级别。

```
WITH RECURSIVE category_path (id, title, lvl) AS
(
    SELECT id, title, 0 lvl
    FROM category
    WHERE parent_id IS NULL
UNION ALL
    SELECT c.id, c.title, cp.lvl + 1
    FROM category_path AS cp JOIN category AS c
    ON cp.id = c.parent_id
)
SELECT * FROM category_path
ORDER BY lvl;
```

如下所示 -

	id	title	lvl
▶	1	Electronics	0
	2	Laptops & PC	1
	5	Cameras & photo	1
	7	Phones & Accessories	1
	3	Laptops	2
	4	PC	2
	6	Camera	2
	8	Smartphones	2
	12	Batteries	2
	13	Headsets	2
	14	Screen Protectors	2
	11	Other Smartphones	3
	9	Android	3
	10	iOS	3

删除节点及其后代

要删除节点及其后代，只需删除节点本身，则所有后代将被删除的 **DELETE CASCADE** 自动删除

例如，要 Laptops & PC 节点及其子节点(Laptops , PC)，请使用以下语句：

```
DELETE FROM category
WHERE
    id = 2;
```

删除节点并提升其后子节点

删除非叶节点并提升其后子节点：

- 首先，将节点的直接子节点的 **parent_id** 更新为新父节点的 **ID**。
- 然后，删除节点。

例如，要删除 Smartphones 节点并其子项，例如 Android , iOS , Other Smartphones 节点：

首先，更新 Smartphones 的所有直接子节点项的 **parent_id**：

```
UPDATE category
SET
    parent_id = 7 -- Phones & Accessories
WHERE
    parent_id = 5; -- Smartphones
```

其次，删除 Smartphones 节点：

```
DELETE FROM category
WHERE
    id = 8;
```

两个语句都应该包含在一个事务中：

```
BEGIN;

UPDATE category
SET
    parent_id = 7
WHERE
    parent_id = 5;

DELETE FROM category
WHERE
    id = 8;

COMMIT;
```

移动子树

要移动子树，只需更新子树的顶级节点的 parent_id 。例如，要移动 Cameras & photo 作为 Phone and Accessories 的子节点，可使用以下语句：

```
UPDATE category
SET
    parent_id = 7
WHERE
    id = 5;
```

在本教程中，您已经学会了如何使用邻接列表模型来管理MySQL中的分层数据

在本教程中，您将学习在数据库中获取MySQL行计数的各种方法。

获取单个表的MySQL行计数

要获取单个表的行计数，可以在SELECT语句中使用COUNT(*)，如下所示：

```
SELECT  
    COUNT(*)  
FROM  
    table_name;
```

例如，要获取示例数据库(yiibaidb)中的 customers 表中的行数，可以使用以下语句：

```
SELECT  
    COUNT(*)  
FROM  
    customers;
```

执行上面查询语句，得到以下结果 -

```
+-----+  
| COUNT(*) |  
+-----+  
|      122 |  
+-----+  
1 row in set (0.01 sec)
```

获取MySQL两个或多个表的行计数

要获取多个表的行数，可以使用UNION运算符组合每个 SELECT 语句返回的结果集。

例如，要在单个查询中获取 customers 和 orders 表的行数，请使用以下语句。

```

SELECT
    'customers' tablename,
    COUNT(*) rows
FROM
    customers
UNION
SELECT
    'orders' tablename,
    COUNT(*) rows
FROM
    orders;

```

获取特定数据库中所有表的MySQL行计数

要获取行计数特定数据库中的所有表，例如 `yiibaidb` 数据，请按照以下步骤：

- 首先，获取数据库中的所有表名
- 第二步，构造一个SQL语句，其中包含由 `UNION` 分隔的所有表的所有 `SELECT COUNT(*) FROM table_name` 语句。
- 第三步，使用准备语句执行SQL语句。

第一步，要获取数据库的所有表名，请从 `information_schema` 数据库中查询如下：

```

SELECT
    table_name
FROM
    information_schema.tables
WHERE
    table_schema = 'yiibaidb'
        AND table_type = 'BASE TABLE';

```

执行上面查询，得到以下结果 -

```
+-----+  
| TABLE_NAME |  
+-----+  
| customers |  
| employees |  
| offices   |  
| orderdetails |  
| orders    |  
| payments   |  
| productlines |  
| products  |  
+-----+  
8 rows in set (0.02 sec)
```

第二步，构造SQL语句，我们使用GROUP_CONCAT和CONCAT函数如下：

```
SELECT  
CONCAT(GROUP_CONCAT(CONCAT('SELECT \'',  
table_name,  
'\' table_name,COUNT(*) rows FROM ',  
table_name)  
SEPARATOR ' UNION '),  
' ORDER BY table_name')  
INTO @sql  
FROM  
table_list;
```

在此查询中，table_list 是第一步中查询结果的表名列表。

以下查询使用第一个查询作为派生表，并以字符串形式返回SQL语句。

```

SELECT
    CONCAT(GROUP_CONCAT(CONCAT('SELECT \'' ,
        table_name,
        '\' table_name,COUNT(*) rows FROM ' ,
        table_name)
        SEPARATOR ' UNION '),
        ' ORDER BY table_name')
INTO @sql
FROM
    (SELECT
        table_name
    FROM
        information_schema.tables
    WHERE
        table_schema = 'yiibaidb'
        AND table_type = 'BASE TABLE') table_list;

```

如果您使用MySQL 8.0+，则可以使用[MySQL CTE](#)(通用表表达式)而不是派生表：

```

WITH table_list AS (
SELECT
    table_name
FROM information_schema.tables
WHERE table_schema = 'yiibaidb' AND
    table_type = 'BASE TABLE'
)
SELECT CONCAT(
    GROUP_CONCAT(CONCAT("SELECT '",table_name,"' table_n
ame,COUNT(*) rows FROM ",table_name) SEPARATOR " UNION "),
    ' ORDER BY table_name'
)
INTO @sql
FROM table_list;

```

第三步，使用[prepare语句](#)执行 @sql 语句，如下所示：

获取行数

```
USE yiibaidb;
PREPARE s FROM @sql;
EXECUTE s;
DEALLOCATE PREPARE s;
```

执行上面查询统计语句，得到以下结果 -

table_name	rows
customers	122
departments	0
employees	23
items	9
offices	7
orderdetails	2998
orders	327
payments	273
productlines	7
products	110
tasks	0
tokens	1

12 rows in set

使用一个查询获取数据库中所有表的**MySQL**行计数

获取数据库中所有表的行计数的快速方法是直接从 `information_schema` 数据库中查询数据：

```
SELECT
    table_name,
    table_rows
FROM
    information_schema.tables
WHERE
    table_schema = 'yiibaidb'
ORDER BY table_rows desc;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT
    table_name,
    table_rows
  FROM
    information_schema.tables
 WHERE
    table_schema = 'yiibaidb'
 ORDER BY table_rows desc;
+-----+-----+
| table_name | table_rows |
+-----+-----+
| orderdetails |      2731 |
| orders |      326 |
| payments |      256 |
| customers |      122 |
| products |      110 |
| employees |       23 |
| items |        9 |
| productlines |       7 |
| offices |       7 |
| tokens |       0 |
| tasks |       0 |
| departments |       0 |
+-----+-----+
12 rows in set
```

此方法有时不准确，因为表中的 `information_schema` 中的行计数和实际行计数不同步。为避免这种情况，您必须在从 `information_schema` 数据库查询行计数之前运行`ANALYZE TABLE`语句。

```
ANALYZE TABLE table_name, ...;
```

在本教程中，您已经学习了各种方法来获取MySQL数据库中一个或多个表的行数。

在本教程中，您将学习如何比较两个表以找到不匹配的记录。

在数据迁移中，我们经常需要比较两个表，以便在一个表中标识另一个表中没有相应记录的记录。

例如，我们有一个新的数据库，其架构与旧数据库不同。我们的任务是将所有数据从旧数据库迁移到新数据库，并验证数据是否正确迁移。

要检查数据，我们必须比较两个表，一个在新数据库中，一个在旧数据库中，并标识不匹配的记录。

假设有两个表：`t1` 和 `t2`。使用以下步骤比较两个表，并确定不匹配的记录：

首先，使用[UNION语句](#)来组合两个表中的行；仅包含需要比较的列。返回的结果集用于比较。

```
SELECT t1.pk, t1.c1
FROM t1
UNION ALL
SELECT t2.pk, t2.c1
FROM t2
```

第二步，根据需要比较的[主键](#)和列分组记录。如果需要比较的列中的值相同，则 `COUNT(*)` 返回 `2`，否则 `COUNT(*)` 返回 `1`。

请参阅以下查询：

```
SELECT pk, c1
FROM
(
    SELECT t1.pk, t1.c1
    FROM t1
    UNION ALL
    SELECT t2.pk, t2.c1
    FROM t2
) t
GROUP BY pk, c1
HAVING COUNT(*) = 1
ORDER BY pk
```

如果比较中涉及的列中的值相同，则不返回任何行。

MySQL 比较两个表的例子

我们来看一个模拟上述步骤的例子。

首先，创建具有相似结构的 2 个表：

```
USE testdb;
CREATE TABLE t1(
    id int auto_increment primary key,
    title varchar(255)
);

CREATE TABLE t2(
    id int auto_increment primary key,
    title varchar(255),
    note varchar(255)
);
```

其次，在 t1 和 t2 表中插入一些数据：

```
INSERT INTO t1(title)
VALUES('row 1'),('row 2'),('row 3');

INSERT INTO t2(title,note)
SELECT title, 'data migration'
FROM t1;
```

第三，比较两个表的 id 和 title 列的值：

```
SELECT id, title
FROM (
    SELECT id, title FROM t1
    UNION ALL
    SELECT id, title FROM t2
)tbl
GROUP BY id, title
HAVING count(*) = 1
ORDER BY id;
```

没有行返回，因为没有不匹配的记录。

第四，在 `t2` 表中插入一行：

```
INSERT INTO t2(title,note)
VALUES('new row 4','new');
```

没有行返回，因为没有不匹配的记录。

第四步，在 `t2` 表中插入一行：...

```
INSERT INTO t2(title,note)
VALUES('new row 4','new');
```

第五步，执行查询以再次比较两个表中的 `title` 列的值。新行是不匹配的行将会返回。

```
mysql> SELECT id, title
    FROM (
        SELECT id, title FROM t1
        UNION ALL
        SELECT id, title FROM t2
    ) tbl
   GROUP BY id, title
  HAVING count(*) = 1
 ORDER BY id;
+----+-----+
| id | title |
+----+-----+
| 4  | new row 4 |
+----+-----+
1 row in set
```

在本教程中，您已经学习了如何根据特定列比较两个表以找到不匹配的记录。

在本教程中，您将学习如何在MySQL中找到一个或多个列的重复值。

在开始之前

由于原因很多，数据库中的重复事件发生很多。查找重复值是使用数据库时必须处理的重要任务之一。

对于演示，我们将创建一个名为 `contacts` 表，其中包含四个列：`id`，`first_name`，`last_name` 和 `email`。

```
USE testdb;

CREATE TABLE contacts (
    id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL
);
```

以下语句将行插入到 `contacts` 表中：

```
INSERT INTO contacts (first_name, last_name, email)
VALUES ('Carine ', 'Schmitt', 'carine.schmitt@qq.com'),
       ('Jean', 'King', 'jean.king@yiibai.com'),
       ('Peter', 'Ferguson', 'peter.ferguson@google.com'),
       ('Janine ', 'Labrune', 'janine.labrune@aol.com'),
       ('Jonas ', 'Bergulfsen', 'jonas.bergulfsen@mac.com'),
       ('Janine ', 'Labrune', 'janine.labrune@aol.com'),
       ('Susan', 'Nelson', 'susan.nelson@qq.com'),
       ('Zbyszek ', 'Piestrzeniewicz', 'zbyszek.piestrzeniewicz@qq
.com'),
       ('Roland', 'Keitel', 'roland.keitel@yahoo.com'),
       ('Julie', 'Murphy', 'julie.murphy@yahoo.com'),
       ('Kwai', 'Lee', 'kwai.lee@google.com'),
       ('Jean', 'King', 'jean.king@qq.com'),
       ('Susan', 'Nelson', 'susan.nelson@qq.comt'),
       ('Roland', 'Keitel', 'roland.keitel@yahoo.com');
```

然后，查询表中的数据如下 -

```
SELECT  
    *  
FROM  
    contacts;
```

执行上面查询，得到以下结果 -

	<code>id</code>	<code>first_name</code>	<code>last_name</code>	<code>email</code>
1	Carine	Schmitt		<code>carine.schmitt@qq.com</code>
2	Jean	King		<code>jean.king@yibai.com</code>
3	Peter	Ferguson		<code>peter.ferguson@google.com</code>
4	Janine	Labrune		<code>janine.labrune@aol.com</code>
5	Jonas	Bergulfsen		<code>jonas.bergulfsen@mac.com</code>
6	Janine	Labrune		<code>janine.labrune@aol.com</code>
7	Susan	Nelson		<code>susan.nelson@qq.com</code>
8	Zbyszek	Piestrzeniewicz		<code>zbyszek.piestrzeniewicz@qq.com</code>
9	Roland	Keitel		<code>roland.keitel@yahoo.com</code>
10	Julie	Murphy		<code>julie.murphy@yahoo.com</code>
11	Kwai	Lee		<code>kwai.lee@google.com</code>
12	Jean	King		<code>jean.king@qq.com</code>
13	Susan	Nelson		<code>susan.nelson@qq.comt</code>
14	Roland	Keitel		<code>roland.keitel@yahoo.com</code>

`14 rows in set`

在 `contacts` 表中，有一些行在 `first_name`，`last_name` 和 `email` 列中具有重复的值，下面来看看如何查询它们。

在一列中找到重复的值

在基于一列的表中找到重复值，则使用以下语句：

```
SELECT
    col,
    COUNT(col)
FROM
    table_name
GROUP BY col
HAVING COUNT(col) > 1;
```

如果表中出现多个值，则该值将被视为重复。在这个语句中，使用 **COUNT** 函数的 **GROUP BY** 子句来计算指定列(**col**)的值。**HAVING** 子句中的条件仅包含值 **count** 大于 **1** 的行，这些行是重复的行。

可以使用此查询在 **contacts** 表中查找具有重复 **email** 的所有行，如下所示：

```
SELECT
    email,
    COUNT(email)
FROM
    contacts
GROUP BY email
HAVING COUNT(email) > 1;
```

以下显示查询的输出：

email	COUNT(email)
janine.labrune@aol.com	2
roland.keitel@yahoo.com	2

2 rows in set

如上查询结果中可以看到，有一些行具有相同的电子邮件。

在多个列中查找重复值

有时，希望基于多个列而不是一个查找重复。在这种情况下，您可以使用以下查询：

```
SELECT
    col1, COUNT(col1),
    col2, COUNT(col2),
    ...
    ...

FROM
    table_name
GROUP BY
    col1,
    col2, ...
HAVING
    (COUNT(col1) > 1) AND
    (COUNT(col2) > 1) AND
    ...
```

只有当列的组合重复时，行才被认为是重复的，所以在 **HAVING** 子句中使用了 **AND** 运算符。

例如，要使用 `first_name`，`last_name` 和 `email` 列中的重复值在 `contacts` 表中查找行，请使用以下查询：

```
SELECT
    first_name, COUNT(first_name),
    last_name, COUNT(last_name),
    email, COUNT(email)
FROM
    contacts
GROUP BY
    first_name ,
    last_name ,
    email
HAVING COUNT(first_name) > 1
    AND COUNT(last_name) > 1
    AND COUNT(email) > 1;
```

执行上面查询后，得到以下输出：

```
+-----+-----+-----+
+-----+-----+
| first_name | COUNT(first_name) | last_name | COUNT(last_name) |
| email       | COUNT(email)      |           |
+-----+-----+-----+
+-----+-----+
| Janine          | 2 | Labrune        | 2 |
| janine.labrune@aol.com |           | 2 |
| Roland          | 2 | Keitel         | 2 |
| roland.keitel@yahoo.com |           | 2 |
+-----+-----+-----+
+-----+-----+
2 rows in set
```

在本教程中，您已经学会了如何根据MySQL中一个或多个列的值来找到重复的行。

本教程将向您展示在MySQL中删除重复行的各种方法。

在上一个教程中，我们向您展示了如何在表中[找到重复的值](#)。当确定了表中有重复的行，您可能需要删除它们来清理这些不必要的数据。

准备示例数据

以下脚本创建 contacts 表，并将示例数据[插入到用于演示的 contacts 表中](#)。

```
USE testdb;
DROP TABLE IF EXISTS contacts;

CREATE TABLE contacts (
    id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) DEFAULT NULL,
    last_name VARCHAR(50) DEFAULT NULL,
    email VARCHAR(255) NOT NULL
);

INSERT INTO contacts (first_name, last_name, email)
VALUES ('Carine ', 'Schmitt', 'carine.schmitt@yiibai.com'),
       ('Jean', 'King', 'jean.king@gmail.com'),
       ('Peter', 'Ferguson', 'peter.ferguson@google.com'),
       ('Janine ', 'Labrune', 'janine.labrune@qq.com'),
       ('Jonas ', 'Bergulfsen', 'jonas.bergulfsen@mac.com'),
       ('Janine ', 'Labrune', 'janine.labrune@qq.com'),
       ('Susan', 'Nelson', 'susan.nelson@qq.com'),
       ('Zbyszek ', 'Piestrzeniewicz', 'zbyszek.piestrzeniewicz@at
t.com'),
       ('Roland', 'Keitel', 'roland.keitel@yahoo.com'),
       ('Julie', 'Murphy', 'julie.murphy@yahoo.com'),
       ('Kwai', 'Lee', 'kwai.lee@google.com'),
       ('Jean', 'King', 'jean.king@qq.com'),
       ('Susan', 'Nelson', 'susan.nelson@qq.com'),
       ('Roland', 'Keitel', 'roland.keitel@yahoo.com');
```

执行[DELETE](#)语句后，可以执行此脚本来重新创建测试数据。

以下查询返回 contacts 表中的重复 email 值：

```
SELECT
    email, COUNT(email)
FROM
    contacts
GROUP BY email
HAVING COUNT(email) > 1;
```

执行上面查询语句，得到以下结果 -

email	COUNT(email)
janine.labrune@qq.com	2
roland.keitel@yahoo.com	2
susan.nelson@qq.com	2

3 rows in set

可以看到，表中有重复 email 行记录。

使用**DELETE JOIN**语句删除重复的行

MySQL为您提供可用于快速删除重复行的**DELETE JOIN**语句。

以下语句删除重复的行并保持最高的ID：

```
DELETE t1 FROM contacts t1
    INNER JOIN
        contacts t2
    WHERE
        t1.id < t2.id AND t1.email = t2.email;

Query OK, 3 rows affected
```

如上所示，有 3 行记录已被删除。我们再次执行查找重复的电子邮件的查询：

```
SELECT
    email, COUNT(email)
FROM
    contacts
GROUP BY email
HAVING COUNT(email) > 1;
```

该查询返回一个空集合，这意味着重复的行已被删除。

我们来查询验证 `contacts` 表中的数据：

```
SELECT
    *
FROM
    contacts;
```

ID 为 4，7 和 9 的行记录已被删除。

如果要删除重复的行并保留最低的 ID，则可以使用以下语句：

```
DELETE t1 FROM contacts t1
INNER JOIN
    contacts t2
WHERE
    t1.id > t2.id AND t1.email = t2.email;
```

可以再次执行创建 `contacts` 表的脚本并测试此查询，以下输出显示删除重复行后的 `contacts` 表的数据。

使用直接表删除重复的行

以下是使用直接表删除重复行的步骤：

- 创建一个新表，其结构与要删除重复行的原始表相同。
- 将原始表中的不同行插入直接表。
- 删除原始表并将直接表重命名为原始表。

以下查询说明了以下步骤：

步骤1 -

```
CREATE TABLE source_copy FROM source;
```

步骤2 -

```
INSERT INTO source_copy
SELECT * FROM source
GROUP BY col; -- column that has duplicate values
```

步骤3 -

```
DROP TABLE source;
ALTER TABLE source_copy RENAME TO source;
```

例如，以下语句从 contacts 表中删除具有重复电子邮件(email)的行记录：

```
-- step 1
CREATE TABLE contacts_temp
LIKE contacts;

-- step 2
INSERT INTO contacts_temp(email) SELECT email FROM contacts GROUP BY email;

-- step 3
DROP TABLE contacts;

ALTER TABLE contacts_temp
RENAME TO contacts;
```

在本教程中，您已经学习了如何使用 DELETE JOIN 语句或直接表删除MySQL中的重复行。

本教程将向您介绍MySQL UUID，并演示如何将其用作表的主键(PK)，并讨论将其用作主键的优缺点。

MySQL UUID简介

UUID 代表通用唯一标识符。UUID 是基于”[RFC 4122](#)“通用唯一标识符(UUID)URN命名空间”)定义的。

UUID 被设计为在空间和时间全球独一无二的数字。预期两个UUID值是不同的，即使它们在两个独立的服务器上生成。

在MySQL中，UUID 值是一个 128 位的数字，表示为以下格式的十五进制数字的 utf8 字符串：

```
aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee
```

要生成 UUID 值，请使用 `UUID()` 函数，如下所示：

```
UUID()
```

`UUID()` 函数返回符合 [RFC 4122](#) 中描述的 UUID 版本1的 UUID 值。

例如，以下语句使用 `UUID()` 函数来生成 UUID 值：

```
mysql> SELECT UUID();
+-----+
| UUID() |
+-----+
| 9d6212cf-72fc-11e7-bdf0-f0def1e6646c |
+-----+
1 row in set
```

MySQL UUID与自动递增INT作为主键

优点

使用 `UUID` 作为 [主键](#)具有以下优点：

- UUID值在表，数据库甚至在服务器上都是唯一的，允许您从不同数据库合并行或跨服务器分发数据库。
- UUID值不会公开有关数据的信息，因此它们在URL中使用更安全。例如，如果 `ID` 为 `10` 的客户通过URL：<http://www.example.com/customers/10/> 访问他的帐户，那么很容易猜到有一个客户 `11`, `12` 等，这可能是攻击的目标。
- 可以在避免往返数据库服务器的任何地方生成 UUID 值。它也简化了应用程序中的逻辑。例如，要将数据插入到父表和子表中，必须首先插入父表，获取生成的 `id`，然后将数据插入到子表中。通过使用 `UUID`，可以生成父表的主键值，并在事务中同时在父表和子表中插入行。

缺点

除了优势之外，UUID值也有一些缺点：

- 存储UUID值(16字节)比整数(4字节)或甚至大整数(8字节)占用更多的存储空间。
- 调试似乎更加困难，想像一下 `WHERE id = '9d6212cf-72fc-11e7-bdf0-f0def1e6646c'` 和 `WHERE id = 10` 那个舒服一点？
- 使用UUID值可能会导致性能问题，因为它们的大小和没有被排序。

MySQL UUID解决方案

在MySQL中，可以以紧凑格式(`BINARY`)存储UUID值，并通过以下功能显示人机可读格式(`VARCHAR`)：

- `UUID_TO_BIN`
- `BIN_TO_UUID`
- `IS_UUID`

请注意，`UUID_TO_BIN()`，`BIN_TO_UUID()` 和 `IS_UUID()` 函数仅在MySQL 8.0或更高版本中可用。

`UUID_TO_BIN()` 函数将UUID从人类可读格式(`VARCHAR`)转换成用于存储的紧凑格式(`BINARY`)格式，并且 `BIN_TO_UUID()` 函数将UUID从紧凑格式(`BINARY`)转换为人类可读格式(`VARCHAR`)。

如果参数是有效的字符串格式 `UUID`，`IS_UUID()` 函数将返回 `1`。如果参数不是有效的字符串格式 `UUID`，则 `IS_UUID` 函数返回 `0`，如果参数为 `NULL`，则 `IS_UUID()` 函数返回 `NULL`。

以下是MySQL中有效的字符串格式 UUID :

```
aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee
aaaaaaaaabbbbccccdddeeeeeeeeeeee
{aaaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee}
```

MySQL UUID示例

我们来看一下使用 `UUID` 作为主键的例子。

以下语句[创建](#)一个名为 `customers` 的新表：

```
USE testdb;
CREATE TABLE customers (
    id BINARY(16) PRIMARY KEY,
    name VARCHAR(255)
);
```

要将 `UUID` 值插入到 `id` 列中，可以使用 `UUID()` 和 `UUID_TO_BIN()` 函数，如下所示：

```
INSERT INTO customers(id, name)
VALUES(UUID_TO_BIN(UUID()), 'John Doe'),
      (UUID_TO_BIN(UUID()), 'Will Minsu'),
      (UUID_TO_BIN(UUID()), 'Mary Jane');
```

要从UUID列查询数据，可以使用 `BIN_TO_UUID()` 函数将二进制格式转换为可读取的格式：

```
SELECT
    BIN_TO_UUID(id) id,
    name
FROM
    customers;
```

在本教程中，您已经了解了MySQL UUID以及如何将其用于主键列。

在本教程中，您将学习如何使用 `CREATE TABLE` 和 `SELECT` 语句在同一数据库或从一个数据库复制表。

MySQL将表复制到新表

将数据从现有表复制到新的数据，在某些情况下非常有用，例如备份数据和复制生产数据进行测试。

要将数据从表复制到新表，请使用`CREATE TABLE`和`SELECT`语句，如下所示：

```
CREATE TABLE new_table
SELECT col1, col2, col3
FROM
existing_table;
```

首先，MySQL使用`CREATE TABLE`语句中指定的名称创建一个新表。新表的结构由 `SELECT` 语句的结果集定义。然后，MySQL将来自 `SELECT` 语句的数据填充到新表中。

要将部分数据从现有表复制到新表中，请在 `SELECT` 语句中使用`WHERE`子句，如下所示：

```
CREATE TABLE new_table
SELECT col1, col2, col3
FROM
existing_table
WHERE
conditions;
```

在创建之前，检查您要创建的表是否已存在是非常重要的。为此，您可以在 `CREATE TABLE` 语句中使用 `IF NOT EXIST` 子句。将数据从现有表复制到新的表的完整命令如下：

```
CREATE TABLE new_table
SELECT col1, col2, col3
FROM
existing_table
WHERE
conditions;
```

请注意，上面的声明只是复制表及其数据。它不会复制与表关联的其他数据库对象，如索引，主键约束，外键约束，触发器等。

要从表中复制数据以及表的所有依赖对象，请使用以下语句：

```
CREATE TABLE IF NOT EXISTS new_table LIKE existing_table;

INSERT new_table
SELECT * FROM existing_table;
```

上面查询中，需要执行两个语句。第一个语句通过复制 `existing_table` 表来创建一个新表 `new_table`。第二个语句将现有 `existing_table` 表中的数据插入到 `new_table` 中。

MySQL拷贝表示例

以下语句将数据从 `office` 表复制到示例数据(`yiibaidb`)库中指定名为 `offices_bk` 的新表。

```
USE yiibaidb;
CREATE TABLE IF NOT EXISTS offices_bk
SELECT * FROM
offices;
```

可以通过查询 `office_bk` 表中的数据来验证副本，如下所示：

```
SELECT  
    *  
FROM  
    offices_bk;
```

执行上面查询，得到以下结果 -

复制表数据

officeCode	city		phone		addressLine1
	addressLine2	state	country	postalCode	territory
1	San Francisco Suite 300	CA	USA	94080	Market Street
2	Boston Suite 102	MA	USA	02107	Court Plaza
3	NYC apt. 5A	NY	USA	10022	East 53rd
4	Paris NULL	NULL	France	75017	EMEA
5	Beijing NULL	BJ	China	110000	NA
6	Sydney Avenue Floor #2	NULL	Australia	NSW 2010	APA
7	London Level 7	NULL	UK	EC2N 1HN	EMEA

如果我们想仅复制在美国(USA)办公室，可以将 WHERE 子句添加到 SELECT 语句中，如下所示：

```
USE yiibaidb;
CREATE TABLE IF NOT EXISTS offices_usa
SELECT *
FROM
    offices
WHERE
    country = 'USA';
```

验证上面查询执行，得到以下结果 -

```
SELECT *
FROM
    offices
WHERE
    country = 'USA';
+-----+-----+-----+-----+
-----+-----+-----+-----+
 | officeCode | city           | phone          | addressLine1
 | addressLine2 | state          | country        | postalCode   | territory |
+-----+-----+-----+-----+
-----+-----+-----+-----+
 | 1           | San Francisco | +1 650 219 4782 | 100 Market Stre
et | Suite 300    | CA             | USA            | 94080        | NA         |
+-----+-----+-----+-----+
 | 2           | Boston         | +1 215 837 0825 | 1550 Court Plac
e | Suite 102    | MA             | USA            | 02107        | NA         |
+-----+-----+-----+-----+
 | 3           | NYC            | +1 212 555 3000 | 523 East 53rd S
treet | apt. 5A       | NY             | USA            | 10022        | NA         |
+-----+-----+-----+-----+
-----+-----+-----+-----+
3 rows in set
```

假设不仅要复制数据，还要复制与 `offices` 表相关联的所有数据库对象，我们使用以下语句：

```
CREATE TABLE offices_dup LIKE offices;  
  
INSERT office_dup  
SELECT * FROM offices;
```

MySQL 复制表到另一个数据库

有时，您要将表复制到其他数据库。在这种情况下，可使用以下语句：

```
CREATE TABLE destination_db.new_table  
LIKE source_db.existing_table;  
  
INSERT destination_db.new_table  
SELECT *  
FROM source_db.existing_table;
```

第一个语句通过从源数据库(`source_db`)复制现有表(`existing_table`)到目标数据库(`destination_db`)中创建一个新表 `new_table` 。

第二个语句将数据从源数据库中的现有(`existing_table`)表复制到目标数据库中的新表。

下面来看看下面的例子。

首先，我们使用以下语句创建一个名为 `testdb` 的数据库：

```
CREATE DATABASE IF NOT EXISTS testdb;
```

其次，通过将其结构从示例数据库(`yiibaidb`)中的 `offices` 表复制出来，在 `testdb` 中创建了 `offices` 表。

```
CREATE TABLE testdb.offices LIKE yiibaidb.offices;
```

第三，我们将数据从 `yiibaidb.offices` 表复制到 `testdb.offices` 表中。

```
INSERT testdb.offices  
SELECT *  
FROM yiibaidb.offices;
```

我们来验证 `testdb.offices` 表中的数据。

```
SELECT  
*  
FROM  
testdb.offices;
```

执行上面查询语句，得到以下结果 -

复制表数据

```
mysql> SELECT
    *
FROM
    testdb.offices;
+-----+-----+-----+-----+-----+-----+
| officeCode | city           | phone          | addressLine1 |
|             | addressLine2   | state          | country       | postalCode   | territory   |
+-----+-----+-----+-----+-----+-----+
| 1          | San Francisco | +1 650 219 4782 | 100 Market Street | | |
|             | Suite 300     | CA             | USA            | 94080        | NA          |
| 2          | Boston         | +1 215 837 0825 | 1550 Court Place |
|             | Suite 102     | MA             | USA            | 02107        | NA          |
| 3          | NYC            | +1 212 555 3000 | 523 East 53rd Street |
|             | apt. 5A       | NY             | USA            | 10022        | NA          |
| 4          | Paris          | +33 14 723 4404 | 43 Rue Jouffroy Dabbans |
|             | NULL           | NULL           | France         | 75017        | EMEA        |
| 5          | Beijing         | +86 33 224 5000 | 4-1 Haidian Area |
|             | NULL           | BJ             | China          | 110000       | NA          |
| 6          | Sydney          | +61 2 9264 2451 | 5-11 Wentworth Avenue |
|             | Floor #2      | NULL           | Australia     | NSW 2010     | APA          |
| 7          | London          | +44 20 7877 2041 | 25 Old Broad Street |
|             | Level 7       | NULL           | UK             | EC2N 1HN     | EME          |
+-----+-----+-----+-----+-----+-----+
7 rows in set
```

在本教程中，我们向您展示了将数据库中的表和从一个数据库复制到另一个数据库的各种技术。

在本教程中，您将学习如何在SQL语句中使用MySQL用户定义的变量。

MySQL用户定义变量简介

有时候，您希望将值从SQL语句传递给另一个SQL语句。为此，您可将该值存储在第一个语句中的MySQL用户定义的变量中，并在随后的语句中引用它。

要创建用户定义的变量，请使用格式 `@variable_name`，其中 `variable_name` 由字母数字字符组成。用户自定义变量的最大长度为64个字符 (MySQL 5.7.5之前的版本)

用户定义的变量不区分大小写。这意味着 `@id` 和 `@ID` 是一样的。

可以将用户定义的变量分配给某些数据类型，例如整数，浮点，小数，字符串或NULL。

由一个客户端定义的用户定义的变量不被其他客户端看到。换句话说，用户定义的变量是特定于会话的。

注意，用户定义的变量是MySQL特定于SQL标准的扩展。它们在其他数据库系统中可能不可用。

MySQL 变量赋值

有两种方法可以将值分配给用户定义的变量。

第一种方法是使用 `SET` 语句如下：

```
SET @variable_name := value;
```

您可以使用 `:=` 或 `=` 作为 `SET` 语句中的赋值运算符。例如，该语句将数字 `100` 分配给变量 `@counter`。

```
SET @counter := 100;
```

为变量分配值的第二种方法是使用SELECT语句。在这种情况下，您必须使用 `:=` 赋值运算符，因为在 `SELECT` 语句中，MySQL将 `=` 运算符视为相等运算符。

```
SELECT @variable_name := value;
```

在赋值之后，可以使用后续语句中允许表达式的变量，例如，在[WHERE子句](#)，[INSERT](#)或[UPDATE](#)语句中。

MySQL 变量示例

以下语句在 `products` 表中查询获得最昂贵的产品，并将价格分配给用户定义的变量 `@msrp`：

```
SELECT
    @msrp:=MAX(msrp)
FROM
    products;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
    @msrp:=MAX(msrp)
FROM
    products;
+-----+
| @msrp:=MAX(msrp) |
+-----+
| 214.30           |
+-----+
1 row in set

mysql> select @msrp as max_price ;
+-----+
| max_price |
+-----+
| 214.30   |
+-----+
1 row in set
```

以下语句使用 `@msrp` 变量来查询最昂贵产品的信息。

```

SELECT
    productCode, productName, productLine, msrp
FROM
    products
WHERE
    msrp = @msrp;

```

有时候，您想要在表中插入一行，获取最后一个插入ID，并使用将此ID作一项数据插入另一个表。在这种情况下，您可以使用用户定义的变量来存储 `AUTO_INCREMENT` 列生成的最新 ID，如下所示。

```
SELECT @id:=LAST_INSERT_ID();
```

用户定义的变量只能保存一个值。如果 `SELECT` 语句返回多个值，则变量将获取结果中最后一行的值。

```

SELECT
    @buyPrice:=buyprice
FROM
    products
WHERE
    buyprice > 95
ORDER BY buyprice;

```

执行上面语句，得到以下结果 -

@buyPrice:=buyprice
95.34
95.59
98.30
98.58
101.51
103.42

6 rows in set

接下来，查询上面变量(`@buyprice`)的值，结果如下所示 -

```
mysql> SELECT @buyprice;
+-----+
| @buyprice |
+-----+
| 103.42 |
+-----+
1 row in set
```

在本教程中，我们向您展示了如何使用SQL语句中的MySQL变量在会话内的语句之间传递数据。

在本教程中，您将学习如何使用MySQL生成的列来存储从表达式或其他列计算的数据。

MySQL生成列简介

创建新表时，可以在`CREATE TABLE`语句中指定表列。然后，使用`INSERT`，`UPDATE`和`DELETE`语句直接修改表列中的数据。

MySQL 5.7引入了一个名为生成列的新功能。它之所以叫作生成列，因为此列中的数据是基于预定义的表达式或从其他列计算的。

例如，假设有以下结构的一个`contacts`表：

```
CREATE TABLE IF NOT EXISTS contacts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
);
```

要获取联系人的全名，请使用`CONCAT()`函数，如下所示：

```
SELECT
    id, CONCAT(first_name, ' ', last_name), email
FROM
    contacts;
```

这不是最优的查询。

通过使用MySQL生成的列，可以重新创建`contacts`表，如下所示：

```

DROP TABLE IF EXISTS contacts;

CREATE TABLE contacts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    fullname varchar(101) GENERATED ALWAYS AS (CONCAT(first_name,
    ' ', last_name)),
    email VARCHAR(100) NOT NULL
);

```

GENERATED ALWAYS as(expression) 是创建生成列的语法。

要测试“全名”列，请在 contacts 表中插入一行。

```

INSERT INTO contacts(first_name, last_name, email)
VALUES('john', 'doe', 'john.doe@yiibai.com');

```

现在，可以从 contacts 表中查询数据。

当从 contacts 表中查询数据时， fullname 列中的值将立即计算。

MySQL提供了两种类型的生成列：存储和虚拟。每次读取数据时，虚拟列都将在运行中计算，而存储的列在数据更新时被物理计算和存储。

基于此定义，上述示例中的 fullname 列是虚拟列。

MySQL生成列的语法

定义生成列的语法如下：

```

column_name data_type [GENERATED ALWAYS] AS (expression)
[VIRTUAL | STORED] [UNIQUE [KEY]]

```

首先，指定列名及其数据类型。

接下来，添加 GENERATED ALWAYS 子句以指示列是生成的列。

然后，通过使用相应的选项来指示生成列的类型：`VIRTUAL` 或 `STORED`。默认情况下，如果未明确指定生成列的类型，MySQL将使用 `VIRTUAL`。

之后，在 `AS` 关键字后面的大括号内指定表达式。该表达式可以包含文字，内置函数，无参数，操作符或对同一表中任何列的引用。如果你使用一个函数，它必须是标量和确定性的。

最后，如果生成的列被存储，可以为它定义一个唯一约束。

MySQL 存储列示例

我们来看一下[示例数据库\(yiibaidb\)](#)中的 `products` 表。

```
mysql> desc products;
+-----+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default | Ex |
+-----+-----+-----+-----+-----+
| tra           | varchar(15) | NO   | PRI |          |    |
+-----+-----+-----+-----+-----+
| productCode   | varchar(15) | NO   | PRI |          |    |
| productName   | varchar(70)  | NO   |      | NULL    |    |
| productLine   | varchar(50)  | NO   | MUL |          |    |
| productScale  | varchar(10)  | NO   |      | NULL    |    |
| productVendor | varchar(50)  | NO   |      | NULL    |    |
| productDescription | text | NO   |      | NULL    |    |
| quantityInStock | smallint(6) | NO   |      | NULL    |    |
| buyPrice      | decimal(10,2) | NO   |      | NULL    |    |
| MSRP          | decimal(10,2) | NO   |      | NULL    |    |
+-----+-----+-----+-----+-----+
9 rows in set
```

使用 `quantityInStock` 和 `buyPrice` 列的数据，通过以下表达式计算每个 `SKU` 的股票值：

```
quantityInStock * buyPrice
```

但是，可以使用以下 `ALTER TABLE ... ADD COLUMN` 语句将名为 `stock_value` 的存储的生成列添加到 `products` 表：

```
ALTER TABLE products
ADD COLUMN stockValue DOUBLE
GENERATED ALWAYS AS (buyprice*quantityinstock) STORED;
```

通常，`ALTER TABLE` 语句需要完整的表重建，因此，如果更改大表是耗时的。但是，虚拟列并非如此。

现在，我们可以直接从 `products` 表中查询库存值。

```
SELECT
    productName, ROUND(stockValue, 2) AS stock_value
FROM
    products;
```

执行上面查询语句，得到以下结果 -

productName	stock_value
1969 Harley Davidson Ultimate Chopper	387209.73
1952 Alpine Renault 1300	720126.90
1996 Moto Guzzi 1100i	457058.75
2003 Harley Davidson Eagle Drag Bike	508073.64
1972 Alfa Romeo GTA	278631.36
1962 Lancia Delta 16V	702325.22
1968 Ford Mustang	6483.12
***** 省略了一大波数据 *****	
The Queen Mary	272869.44
American Airlines: MD-11s	319901.40
Boeing X-32A JSF	159163.89
Pont Yacht	13786.20
+-----+	
110 rows in set	

在本教程中，我们向您介绍了MySQL生成的列以存储从表达式或其他列计算的数据。

在本教程中，我们将介绍如何使用自连接技术来比较同一个表中的连续行。

假设您有一个名为 `inventory` 的表，其中包含由`CREATE TABLE`语句定义的结构，如下所示：

```
USE testdb;
CREATE TABLE inventory(
    id INT AUTO_INCREMENT PRIMARY KEY,
    counted_date DATE NOT NULL,
    item_no VARCHAR(20) NOT NULL,
    qty int(11) NOT NULL
);
```

在库存(`inventory`)表中：

- `id` 是自动增量列。
- `counted_date` 是计数日期。
- `item_no` 是发布到广告资源的商品代码。
- `qty` 是库存中累计的现货数量。

以下是 `inventory` 表的示例数据：

```
INSERT INTO inventory(counted_date, item_no, qty)
VALUES ('2017-10-01', 'A', 20),
       ('2017-10-01', 'A', 30),
       ('2017-10-01', 'A', 45),
       ('2017-10-01', 'A', 80),
       ('2017-10-01', 'A', 100);
```

如果想知道每个物品每天收到的物品数量，需要将特定日期的现有数量与前一天进行比较。

换句话说，在 `inventory` 表中，需要将一行与其连续行进行比较以找出差异。

在MySQL中，可以使用[自连接技术](#)来比较连续的行，如以下查询：

```

SELECT
    g1.item_no,
    g1.counted_date from_date,
    g2.counted_date to_date,
    (g2.qty - g1.qty) AS receipt_qty
FROM
    inventory g1
        INNER JOIN
    inventory g2 ON g2.id = g1.id + 1
WHERE
    g1.item_no = 'A';

```

执行上面查询语句，得到以下结果 -

item_no	from_date	to_date	receipt_qty
A	2017-10-01	2017-10-01	10
A	2017-10-01	2017-10-01	15
A	2017-10-01	2017-10-01	35
A	2017-10-01	2017-10-01	20

4 rows in set

`INNER JOIN`子句 `g2.id = g1.id + 1` 中的条件允许您将当前行与 `inventory` 表中的下一行进行比较，当然，假设 `id` 列中没有间隙。

如果无法避免差距，您可以创建一个附加列，例如 `seq` 来维护行的顺序，以便应用此技术。

在本教程中，您已经学会了如何使用自连接技术来比较同一个表中的连续行。

在本教程中，您将学习如何使用MySQL REGEXP 运算符执行基于正则表达式的复杂搜索。

正则表达式简介

正则表达式是描述搜索模式的特殊字符串。它是一个强大的工具，为您提供一种简洁灵活的方法来识别基于模式的文本字符，例如字符，单词等。

例如，可以使用正则表达式来搜索电子邮件，IP地址，电话号码，社会安全号码或具有特定模式的任何内容。

正则表达式使用其可以由正则表达式处理器解释自己的语法。正则表达式广泛应用于从编程语言到数据库(包括MySQL)大部分平台。

使用正则表达式的优点是，不限于在LIKE运算符中基于具有百分号(%)和下划线(_)的固定模式搜索字符串。使用正则表达式，有更多的元字符来构造灵活的模式。

正则表达式的缩写是 regex 或 regexp 。

MySQL REGEXP 运算符

MySQL适应了Henry Spencer实现的正则表达式。MySQL允许使用 REGEXP 运算符在SQL语句中匹配模式。

下面说明了WHERE子句中 REGEXP 运算符的语法：

```
SELECT
    column_list
FROM
    table_name
WHERE
    string_column REGEXP pattern;
```

此语句执行 string_column 与模式 pattern 匹配。

如果 string_column 中的值与模式 pattern 匹配，则 WHERE 子句中的表达式将返回 1 ，否则返回 0 。

如果 `string_column` 或 `pattern` 为 `NULL`，则结果为 `NULL`。

除了 `REGEXP` 运算符之外，可以使用 `RLIKE` 运算符，这是 `REGEXP` 运算符的同义词。

`REGEXP` 运算符的否定形式是 `NOT REGEXP`。

MySQL REGEXP示例

假设想找出名字以字母 `A`，`B` 或 `C` 开头的产品。可以使用[SELECT语句](#)中的正则表达式如下：

```
SELECT
    productname
FROM
    products
WHERE
    productname REGEXP '^ (A|B|C) '
ORDER BY productname;
```

执行上面查询语句，得到以下结果 -

```
+-----+
| productname |
+-----+
| America West Airlines B757 200 |
| American Airlines: B767 300 |
| American Airlines: MD-11s |
| ATA: B757-300 |
| Boeing X-32A JSF |
| Collectable Wooden Train |
| Corsair F4U (Bird Cage) |
+-----+
7 rows in set
```

该模式允许查询名称以 `A`，`B` 或 `C` 开头的产品。

- 字符 `^` 表示从字符串的开头匹配。
- 字符 `|` 如果无法匹配，则搜索替代方法。

下表说明了正则表达式中一些常用的元字符和构造。

元字符	行为
^	匹配搜索字符串开头处的位置
\$	匹配搜索字符串末尾的位置
.	匹配任何单个字符
[...]	匹配方括号内的任何字符
[^...]	匹配方括号内未指定的任何字符
p1 p2	匹配 p1 或 p2 模式
*	匹配前面的字符零次或多次
+	匹配前一个字符一次或多次
{n}	匹配前几个字符的 n 个实例
{m,n}	从 m 到 n 个前一个字符的实例匹配

要查找名称以 a 开头的产品，您可以在名称开头使用“ ^ ”进行匹配，如下查询语句：

```
SELECT
    productname
FROM
    products
WHERE
    productname REGEXP '^a';
```

执行上面查询语句，得到以下结果 -

```
+-----+
| productname |
+-----+
| American Airlines: B767 300 |
| America West Airlines B757 200 |
| ATA: B757 300 |
| American Airlines: MD 11S |
+-----+
4 rows in set
```

如果要使 `REGEXP` 运算符以区分大小写的方式比较字符串，可以使用 `BINARY` 运算符将字符串转换为二进制字符串。

因为MySQL比较二进制字节逐字节而不是逐字符。这允许字符串比较区分大小写。

例如，以下语句只匹配开头为大写“ C ”的产品名称。

```
SELECT
    productname
FROM
    products
WHERE
    productname REGEXP BINARY '^C';
```

执行上面查询语句，得到以下结果 -

```
+-----+
| productname |
+-----+
| Collectable Wooden Train |
| Corsair F4U ( Bird Cage) |
+-----+
2 rows in set
```

要找到以 `f` 结尾的产品，您可以使用’ `$f` ‘来匹配字符串的末尾。

```
SELECT
    productname
FROM
    products
WHERE
    productname REGEXP 'f$';
```

执行上面查询语句，得到以下结果 -

```
+-----+  
| productname |  
+-----+  
| Boeing X-32A JSF |  
+-----+  
1 row in set
```

要查找其名称包含“`ford`”的产品，请使用以下查询：

```
SELECT  
    productname  
FROM  
    products  
WHERE  
    productname REGEXP 'ford';
```

执行上面查询语句，得到以下结果 -

```
+-----+  
| productname |  
+-----+  
| 1968 Ford Mustang |  
| 1969 Ford Falcon |  
| 1940 Ford Pickup Truck |  
| 1911 Ford Town Car |  
| 1932 Model A Ford J-Coupe |  
| 1926 Ford Fire Engine |  
| 1913 Ford Model T Speedster |  
| 1934 Ford V8 Coupe |  
| 1903 Ford Model A |  
| 1976 Ford Gran Torino |  
| 1940s Ford truck |  
| 1957 Ford Thunderbird |  
| 1912 Ford Model T Delivery Wagon |  
| 1940 Ford Delivery Sedan |  
| 1928 Ford Phaeton Deluxe |  
+-----+  
15 rows in set
```

要查找名称只包含 10 个字符的产品，可以使用' ^ '和' \$ '来匹配产品名称的开头和结尾，并重复 {10} 次任何字符 . ，作为以下查询：

```
SELECT
    productname
FROM
    products
WHERE
    productname REGEXP '^.{10}$';
```

执行上面查询语句，得到以下结果 -

```
+-----+
| productname |
+-----+
| HMS Bounty |
| Pont Yacht |
+-----+
2 rows in set
```

在本教程中，您已学习如何使用具有正则表达式的MySQL REGEXP 运算符查询数据。

在本教程中，您将学习如何在MySQL中模拟 `row_number` 函数。我们将向您展示如何向每行或每组行添加行号。

row_number函数简介

`row_number` 是一个排序函数，返回一行的顺序号，从第一行的 `1` 开始。经常想使用 `row_number` 函数来生成特定的报告。

MySQL不提供像SQL Server，Oracle或PostgreSQL中那样的 `row_number` 函数。幸运的是，MySQL提供了可用于模拟 `row_number` 函数的会话变量。

MySQL row_number - 为每行添加行号

要模拟 `row_number` 函数，必须在查询中使用会话变量。

以下语句从 `employees` 表中获取 5 名员工，并从 `1` 开始为每行添加行号。

```
SET @row_number = 0;

SELECT
    (@row_number:=@row_number + 1) AS num, firstName, lastName
FROM
    employees
LIMIT 5;
```

执行上面语句，得到以下结果 -

```
+-----+-----+-----+
| num | firstName | lastName |
+-----+-----+-----+
| 1   | Diane     | Murphy   |
| 2   | Mary      | Hill     |
| 3   | Jeff      | Firrelli |
| 4   | William   | Patterson |
| 5   | Gerard    | Bondur   |
+-----+-----+-----+
5 rows in set
```

在上述查询语句中：

- 在第一个语句中，定义了一个名为 `row_number` 的变量，并将其值设置为 `0`。`row_number` 是由 `@` 前缀指示的会话变量。
- 在第二个语句中，从 `employees` 表中查询选择了数据，并将 `row_number` 变量的值增加到每行的 `1`。`LIMIT` 子句用于约束一些返回的行，在这种情况下，它被设置为 `5`。

另一种技术是使用会话变量作为派生表，并将其与主表连接。请参阅以下查询：

```
SELECT
    (@row_number:="@row_number + 1") AS num, firstName, lastName
FROM
    employees, (SELECT @row_number:=0) AS t
LIMIT 5;
```

请注意，派生表必须具有自己的别名，以使查询语法正确。

MySQL row_number - 为每个组添加行号

如果要为每个组添加一个行号，并且为每个新组重置它。

我们来看看 `payments` 表：

```
SELECT
    customerNumber, paymentDate, amount
FROM
    payments
ORDER BY customerNumber;
```

执行上面查询语句，得到类似以下结果 -

customerNumber	paymentDate	amount
103	2014-10-19	6066.78
103	2013-06-05	14571.44
103	2014-12-18	1676.14
112	2014-12-17	14191.12
*****省略部分数据*****		
496	2015-05-25	30253.75
496	2013-07-16	32077.44
496	2014-12-31	52166

273 rows in set

假设每个客户要添加一个行号，并且每当客户号码更改时，行号都会被重置。

要实现这一点，可使用两个会话变量，一个用于行号，另一个用于存储旧客户编号，以将其与当前的客户编号进行比较，如下查询语句：

```

SELECT
    @row_number:=CASE
        WHEN @customer_no = customerNumber THEN @row_number + 1
        ELSE 1
    END AS num,
    @customer_no:=customerNumber as CustomerNumber,
    paymentDate,
    amount
FROM
    payments
ORDER BY customerNumber;

```

在查询中使用了 CASE 语句。如果客户号码保持不变，我们增加了 row_number 变量的值，否则将 row_number 变量重置为 1。查询结果如下所示。

row_number 模拟

num	CustomerNumber	paymentDate	amount
*****此处省略了一大波数据*****			
1	103	2014-10-19	6066.78
2	103	2013-06-05	14571.44
3	103	2014-12-18	1676.14
1	112	2014-12-17	14191.12
2	112	2013-06-06	32641.98
3	112	2014-08-20	33347.88
1	114	2013-05-20	45864.03
2	114	2014-12-15	82261.22
3	114	2013-05-31	7565.08
4	114	2014-03-10	44894.74
1	119	2014-11-14	19501.82
2	119	2014-08-08	47924.19
3	119	2015-02-22	49523.67
1	121	2013-02-16	50218.95
1	496	2015-05-25	30253.75
2	496	2013-07-16	32077.44
3	496	2014-12-31	52166

273 rows in set

像 `row_number` 的行号一样，可以使用派生表和交叉连接技术来产生相同的结果。

在本教程中，您将学习从MySQL中的数据库表中查询选择随机记录的各种技术。

有时，必须从表中选择随机记录，例如：

- 在博客中选择一些随机的帖子，并在侧边栏中显示。
- 选择一个随机报价来显示“当天的报价”小部件。
- 在画廊中选择随机图片，并将其用作精选照片。

MySQL 使用 ORDER BY RAND() 选择随机记录

MySQL没有任何内置语句从数据库表中选择随机记录。为了实现这一点，您可以使用 `RAND` 函数。以下查询从数据库表中选择一个随机记录：

```
SELECT
*
FROM
tbl
ORDER BY RAND()
LIMIT 1;
```

让我们更详细地查看上面的查询语句 -

- `RAND()` 函数为表中的每一行生成一个随机值。
- `ORDER BY` 子句通过 `RAND()` 函数生成的随机数对表中的所有行进行排序。
- `LIMIT` 子句选择随机排序的结果集中的第一行。

如果要从数据库表中选择 `N` 个随机记录，则需要如下更改 `LIMIT` 子句：

```
SELECT
*
FROM
table
ORDER BY RAND()
LIMIT N;
```

例如，要在 `customers` 表中查询选择 5 个随机客户，请使用以下查询：

```
SELECT  
    t.customerNumber, t.customerName  
FROM  
    customers AS t  
ORDER BY RAND()  
LIMIT 5;
```

因为是随机，您可能会得到一个不同的结果集。

这种技术与一个小表上非常好。但是对于大表将是非常缓慢的，因为MySQL必须对整个表进行排序以选择随机数。查询的速度也取决于表中的行数。表的行数越多，生成的随机数所花的时间就越多。

MySQL 使用 **INNER JOIN** 子句选择随机记录

该技术要求您要选择随机记录的表具有[自动增量主键](#)字段，并且序列中没有间隙。

以下查询根据主键列生成随机数：

```
SELECT  
    ROUND(RAND() * ( SELECT  
        MAX(id)  
    FROM  
        table)) as id;
```

我们可以通过上述查询返回的结果集[连接](#)表，如下所示：

```
SELECT
    t.*  

FROM
    table AS t
    JOIN
        (SELECT
            ROUND(RAND()) * (SELECT
                MAX(id)
            FROM
                table )) AS id
        ) AS x
WHERE
    t.id >= x.id
LIMIT 1;
```

使用这种技术，您可多次执行查询以获取多个随机行，因为如果增加限制，则查询将仅给出从随机选择的行开始的顺序行。

以下查询从 `customers` 表返回一个随机客户。

```
SELECT
    t.customerNumber, t.customerName
FROM
    customers AS t
    JOIN
        (SELECT
            ROUND(RAND()) * (SELECT
                MAX(customerNumber)
            FROM
                customers)) AS customerNumber
        ) AS x
WHERE
    t.customerNumber >= x.customerNumber
LIMIT 1;
```

MySQL 使用变量选择随机记录

如果表的 `id` 列的值范围为 `1 .. N`，并且范围内没有间隙，则可以使用以下技术：

- 首先，选择 `1..N` 范围内的随机数字。
- 第二步，根据随机数选择记录。

以下语句可以完成此操作：

```
SELECT
    table.* 
FROM
    (SELECT
        ROUND(RAND() * (SELECT
            MAX(id)
        FROM
            table)) random_num,
        @num:=@num + 1
    FROM
        (SELECT @num:=0) AS a, table
    LIMIT N) AS b,
    table AS t
WHERE
    b.random_num = t.id;
```

请注意，用户定义的变量是连接特定的。这意味着这种技术不能与连接池一起使用。此外，主键必须是整数类型，其值必须在没有间隙的序列中。

在本教程中，我们向您展示了从表中选择随机记录的几种技术。

在本教程中，您将学习如何使用各种技术选择数据库表中第 n 个最高记录。

使用 **MAX** 或 **MIN** 函数可以轻松选择数据库表中最高或最低的记录。但是，选择第 n 个最高纪录有点棘手。例如，从 `products` 表中获得第二高价格的产品。

要选择第 n 个最高记录，需要执行以下步骤：

- 首先，得到 n 个最高记录，并按升序排列。第 n 个最高记录是结果集中的最后一个记录。
- 然后，按顺序对结果集进行排序，并获得第一个结果集。

以下是以升序获得第 n 个最高记录的查询：

```
SELECT
*
FROM
    table_name
ORDER BY column_name ASC
LIMIT N;
```

获得第 n 个最高记录的查询如下：

```
SELECT
*
FROM
    (SELECT
        *
    FROM
        table_name
    ORDER BY column_name ASC
    LIMIT N) AS tbl
ORDER BY column_name DESC
LIMIT 1;
```

幸运的是，MySQL为我们提供了限制返回结果集中的行数的**LIMIT子句**。可以重写上述查询如下：

```
SELECT
    *
FROM
    table_name
ORDER BY column_name DESC
LIMIT n - 1, 1;
```

查询返回 $n-1$ 行之后的第一行，以便获得第 n 个最高记录。

获得第n个最高纪录的例子

第一种方法

例如，如果要在 `products` 表中获得第二个最昂贵的产品($n = 2$)，则使用以下查询：

```
SELECT
    productCode, productName, buyPrice
FROM
    products
ORDER BY buyPrice DESC
LIMIT 1 , 1;
```

执行上面查询语句，结果如下：

productCode	productName	buyPrice
S18_2238	1998 Chrysler Plymouth Prowler	101.51

1 row in set

第二种方法

获得第 n 个最高记录的第二种技术是使用[MySQL子查询](#)：

```
SELECT *
FROM table_name AS a
WHERE n - 1 = (
    SELECT COUNT(primary_key_column)
    FROM products b
    WHERE b.column_name > a.column_name)
```

可以使用第一种技术获得相同的结果，以获得第二高价产品作为以下查询：

```
SELECT
    productCode, productName, buyPrice
FROM
    products a
WHERE
    1 = (SELECT
        COUNT(productCode)
    FROM
        products b
    WHERE
        b.buyPrice > a.buyPrice);
```

执行上面查询语句，结果如下：

productCode	productName	buyPrice
S18_2238	1998 Chrysler Plymouth Prowler	101.51

1 row in set

在本教程中，我们向您展示了如何使用MySQL中的 `LIMIT` 子句在数据库表中选择第 `n` 条记录。

在本教程中，我们将向您展示如何重置MySQL中 `AUTO_INCREMENT` 列的自动增量值。

MySQL提供了一个有用的功能，称为自动增量。您可以将自动递增属性分配给表的列，以生成新行的唯一标识。通常，使用表的主键列的自动递增属性。

无论何时在表中插入新行，MySQL会自动将序列值分配给自动增量列。

例如，如果表有 8 行，并插入新行而不指定自动增量列的值，MySQL将自动插入一个 `id` 为 9 的新行。

有时，可能需要重置自动增量列的值，以便插入到表中的第一个记录的标识从特定的数字开始，例如重置为 1 。

在MySQL中，可以以各种方式重置自动增量值。

MySQL重置自动增量值示例

首先，创建一个名为 `tmp` 的表，并将 `AUTO_INCREMENT` 属性分配给 `id` 主键列。

```
USE testdb;
CREATE TABLE tmp (
    id int(11) NOT NULL AUTO_INCREMENT,
    name varchar(45) DEFAULT NULL,
    PRIMARY KEY (id)
);
```

其次，将一些样本数据插入到 `tmp` 表中：

```
INSERT INTO tmp(name)
VALUES('test 1'),
      ('test 2'),
      ('test 3');
```

第三步，查询 `tmp` 表验证插入操作：

```
SELECT * FROM tmp;
```

id	name
1	test 1
2	test 2
3	test 3

3 rows in set

所上查询结果所示，总共有三行，`ID` 列的值分别为：`1`，`2` 和 `3`，现在，来看看如何重置 `ID` 列的自动增量值。

MySQL通过使用**ALTER TABLE**语句重置自动增量值

可以使用**ALTER TABLE**语句重置自动增量值。重置自动增量的值的 `ALTER TABLE` 语句的语法如下：

```
ALTER TABLE table_name AUTO_INCREMENT = value;
```

您可以在 `ALTER TABLE` 子句之后指定表名，并在表达式 `AUTO_INCREMENT = value` 中指定要重置的值。

请注意，该值必须大于或等于自动增量列的当前最大值。

我们删除 `tmp` 表中的 `id` 值为 `3` 的最后一个记录：

```
DELETE FROM tmp WHERE ID = 3;
```

如果插入一行，MySQL将把 `4` 分配给新行的 `id` 列。但是，可以使用 `ALTER TABLE` 语句将由MySQL生成的数字重置为 `3`，如下所示：

```
ALTER TABLE tmp AUTO_INCREMENT = 3;
```

现在，尝试向 `tmp` 表中插入一个新行并从中查询数据以查看效果：

```
INSERT INTO tmp(name)
VALUES ('MySQL example 3');

SELECT
*
FROM
tmp;
```

执行上面查询语句，得到以下结果 -

```
+----+-----+
| id | name |
+----+-----+
| 1  | test 1 |
| 2  | test 2 |
| 4  | MySQL example 3 |
+----+-----+
3 rows in set
```

现在，表中有三行记录，最后一个自动递增值为 3 而不是 4，这是像我们所预期的那样。

MySQL 使用 **TRUNCATE TABLE** 语句重置自动增量值

TRUNCATE TABLE 语句删除表的所有数据，并将 `auto_increment` 值重置为 0。下面说明了 **TRUNCATE TABLE** 语句的语法：

```
TRUNCATE TABLE table_name;
```

通过使用 **TRUNCATE TABLE** 语句，可以删除表的所有数据，并将自动递增值重置为零。

MySQL 使用 **DROP TABLE** 和 **CREATE TABLE** 语句重置自动增量值

您可以使用一对语句：**DROP TABLE** 和 **CREATE TABLE** 来重置自动增量列。

像 **TRUNCATE TABLE** 语句一样，这些语句删除删除表并重新创建它，因此，自动增量的值将重置为 0。

```
DROP TABLE table_name;  
CREATE TABLE table_name(...);
```

在本教程中，您已经学会了如何以各种方式重置MySQL中的自动增量值。第一种方法是最好的，因为它是最简单的方法，没有副作用。

下表说明了 MariaDB 和 MySQL 之间的主要区别：

~	MySQL
开发商	Oracle公司
协议	MySQL
源代码	开源+专有
开发	关闭
合作	有限
存储引擎	InnoDB,MyISAM,BLACKHOLE,CSV,MEMORY,ARCHIVE,MRC
检查约束	No
DEFAULT表达式	No
虚拟列	Yes
动态列	No
角色	Yes
DELETE ... RETURNING	Yes
GIS支持	Yes
ALTER TABLE和LOAD DATA INFILE语句的进度报告	No
表消除	No
SQL管理	MySQL Workbench
监控	MySQL Enterprise Monitor
备份	MySQL Enterprise Backup
SQL公用表表达式(CTE)	Yes(MySQL8.0+)
SQL窗口函数	Yes(MySQL8.0+)
JSON支持	Yes
数据屏蔽	No

加密	MySQL Enterprise Encryption
数据库防火墙	MySQL Enterprise Firewall
审计	MySQL Enterprise Audit
Analytics(分析)	No
分区	MySQL Partitioning
路由	MySQL Router
复制	MySQL Replication

在本教程中，您将学习如何使用MySQL间隔值来执行日期和时间算术运算。

MySQL间隔值简介

MySQL间隔值主要用于日期和时间计算。要创建间隔值，请使用以下表达式：

```
INTERVAL expr unit
```

其次是 INTERVAL 关键字是确定间隔值的 expr ，以及指定间隔单位的单位。例如，要创建 1 天间隔，请使用以下表达式：

```
INTERVAL 1 DAY
```

请注意， INTERVAL 和 UNIT 不区分大小写，因此以下表达式与上述表达式相当：

```
interval 1 day
```

主要使用日期和时间算术的间隔值，如下所示：

```
date + INTERVAL expr unit  
date - INTERVAL expr unit
```

间隔值也用于各种时间函数，如

[DATE_ADD](#)，[DATE_SUB](#)，[TIMESTAMPADD](#) 和 [TIMESTAMPDIFF](#) 。

MySQL定义了 expr 和 unit 的标准格式，如下表所示：

单位(unit)	表达式(expr)
DAY	DAYS
DAY_HOUR	'DAYS HOURS'
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
HOUR	HOURS
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECOND'
HOUR_MINUTE	'HOURS:MINUTES'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
MICROSECOND	MICROSECONDS
MINUTE	MINUTES
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
MONTH	MONTHS
QUARTER	QUARTERS
SECOND	SECONDS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
WEEK	WEEKS
YEAR	YEARS
YEAR_MONTH	'YEARS-MONTHS'

MySQL 间隔示例

以下语句在 2020-01-01 日期上增加1天返回结果为： 2020-01-02 :

```
mysql> SELECT '2020-01-01' + INTERVAL 1 DAY;
+-----+
| '2020-01-01' + INTERVAL 1 DAY |
+-----+
| 2020-01-02 |
+-----+
1 row in set (0.01 sec)
```

如果在涉及DATE或DATETIME值的表达式中使用了间隔值，并且间隔值位于表达式的右侧，则可以使用expr的负值，如以下示例所示：

```
mysql> SELECT '2020-01-01' + INTERVAL -1 DAY;
+-----+
| '2020-01-01' + INTERVAL -1 DAY |
+-----+
| 2019-12-31 |
+-----+
1 row in set
```

以下语句显示如何使用DATE_ADD和DATE_SUB从日期值添加/减去1个月：

```
mysql> SELECT DATE_ADD('2020-01-01', INTERVAL 1 MONTH) 1_MONTH_LATER,
        DATE_SUB('2020-01-01', INTERVAL 1 MONTH) 1_MONTH_BEFORE;
+-----+-----+
| 1_MONTH_LATER | 1_MONTH_BEFORE |
+-----+-----+
| 2020-02-01 | 2019-12-01 |
+-----+-----+
1 row in set
```

以下查询使用TIMESTAMPADD(unit, interval, expression)函数向时间戳值添加30分钟：

```
mysql> SELECT TIMESTAMPADD(MINUTE, 30, '2020-01-01') 30_MINUTES_LATER;
+-----+
| 30_MINUTES_LATER |
+-----+
| 2020 01 01 00:30:00 |
+-----+
1 row in set
```

MySQL 间隔实例

我们创建一个名为 `memberships` 的新表，用于演示：

```
USE testdb;
CREATE TABLE memberships (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(355) NOT NULL,
    plan VARCHAR(255) NOT NULL,
    expired_date DATE NOT NULL
);
```

在 `memberships` 表中，`expired_date` 列存储每个会员的会员资格到期日。

以下语句将一些行插入到 `memberships` 表中。

```
INSERT INTO memberships(email, plan, expired_date)
VALUES('john.doe@yiibai.com', 'Gold', '2018-07-13'),
      ('jane.minsu@yiibai.com', 'Platinum', '2018-07-10'),
      ('david.corp@yiibai.com', 'Silver', '2018-07-15'),
      ('julia.william@yiibai.com', 'Gold', '2018-07-20'),
      ('peter.drucker@yiibai.com', 'Silver', '2018-07-08');
```

假设今天是 `2018-07-06`，您可以使用以下语句查询在 `7` 天内会员资格已过期的会员：

```

SELECT
    email,
    plan,
    expired_date,
    DATEDIFF(expired_date, '2018-07-06') remaining_days
FROM
    memberships
WHERE
    '2018-07-06' BETWEEN DATE_SUB(expired_date, INTERVAL 7 DAY)
AND expired_date;

```

执行上面查询语句后，得到以下结果 -

email	plan	expired_date	remaining_days
john.doe@yiibai.com	Gold	2018-07-13	7
jane.minsu@yiibai.com	Platinum	2018-07-10	4
peter.drucker@yiibai.com	Silver	2018-07-08	2

3 rows in set

在此查询中，我们使用 `DATE_SUB` 函数将间隔值(`INTERVAL 7 DAY`)指定的过期日期减去 7 天。

在本教程中，您已经学习了如何使用MySQL间隔值进行日期和时间算术。

在本教程中，您将学习如何使用MySQL `NULL` 值。此外，您将学习一些有用的函数来有效地处理 `NULL` 值。

如果不能理解和使用数据库中 `NULL` 值，那么可以认为您的数据库学习最多算刚入门水平。

MySQL NULL值简介

在MySQL中，`NULL` 值表示一个未知值。`NULL` 值不同于 `0` 或空字符串 `''`。

`NULL` 值不等于它自身。如果将 `NULL` 值与另一个 `NULL` 值或任何其他值进行比较，则结果为 `NULL`，因为一个不知道是什么的值(`NULL` 值)与另一个不知道是什么的值(`NULL` 值)比较，其值当然也是一个不知道是什么的值(`NULL` 值)。

通常，使用 `NULL` 值来表示数据丢失，未知或不适用的情况。例如，潜在客户的电话号码可能为 `NULL`，并且可以稍后添加。

创建表时，可以通过使用 `NOT NULL` 约束来指定列是否接受 `NULL` 值。

例如，以下语句用于创建一张 `leads` 表：

```
USE testdb;
CREATE TABLE leads (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    source VARCHAR(255) NOT NULL,
    email VARCHAR(100),
    phone VARCHAR(25)
);
```

因此，`id` 是主键列，它不接受任何 `NULL` 值。

`first_name`，`last_name` 和 `source` 列使用 `NOT NULL` 约束，因此，不能在这些列中插入任何 `NULL` 值，而 `email` 和 `phone` 列则可接受 `NULL` 值。

您可以在`INSERT`语句中使用 `NULL` 值来指定数据丢失。例如，以下语句将一行插入到线索表中。因为电话号码丢失，所以使用 `NULL` 值。

```
INSERT INTO leads(first_name, last_name, source, email, phone)
VALUES('John', 'Doe', 'Web Search', 'john.doe@yiibai.com', NULL);
```

因为 `email` 列的默认值为 `NULL`，可以按照以下方式在 `INSERT` 语句中省略电子邮件：

```
INSERT INTO leads(first_name, last_name, source, phone)
VALUES('Lily', 'Bush', 'Cold Calling', '(408)-555-1234'),
('David', 'William', 'Web Search', '(408)-888-6789');
```

UPDATE语句中的MySQL SET NULL值

要将列的值设置为 `NULL`，可以使用赋值运算符(`=`)。例如，要将 `David William` 的手机(`phone`)更新为 `NULL`，请使用以下 `UPDATE` 语句：

```
UPDATE leads
SET
    phone = NULL
WHERE
    id = 3;
```

MySQL ORDER BY为NULL

如果使用`ORDER BY`子句按升序对结果集进行排序，则MySQL认为 `NULL` 值低于其他值，因此，它会首先显示 `NULL` 值。

以下查询语句按照电话号码(`phone`)升序排列。如下所示 -

```
SELECT
*
FROM
leads
ORDER BY phone;
```

执行上面查询语句，结果如下 -

	id	first_name	last_name	source	email
			phone		
1	John	Doe		Web Search	john.doe@yiibai.c
2	David	William		Web Search	NULL
3	Lily	Bush	(408) - 555 - 1234	Cold Calling	NULL

如果使用 `ORDER BY DESC` , `NULL` 值将显示在结果集的最后。请参阅以下示例：

```
SELECT
*
FROM
leads
ORDER BY phone DESC;
```

执行上面查询语句，结果如下 -

<code>id</code>	<code>first_name</code>	<code>last_name</code>	<code>source</code>	<code>email</code>
	<code>phone</code>			
2	Lily	Bush	Cold Calling	NULL
	(408) 555-1234			
1	John	Doe	Web Search	john.doe@yiibai.c
om	NULL			
3	David	William	Web Search	NULL
	NULL			

3 rows in set

要在查询中测试 `NULL`，可以在 `WHERE` 子句中使用 `IS NULL` 或 `IS NOT NULL` 运算符。

例如，要获得尚未提供电话号码的潜在客户，请使用 `IS NULL` 运算符，如下所示：

```
SELECT
*
FROM
    leads
WHERE
    phone IS NULL;
```

执行上面查询语句，结果如下 -

```

+-----+-----+-----+-----+
| id | first_name | last_name | source      | email
| phone |
+-----+-----+-----+-----+
| 1 | John       | Doe        | Web Search | john.doe@yibai.com
| NULL |
| 3 | David      | William    | Web Search | NULL
| NULL |
+-----+-----+-----+-----+
| 2 rows in set

```

可以使用 `IS NOT` 运算符来获取所有提供电子邮件地址的潜在客户。

```

SELECT
*
FROM
leads
WHERE
email IS NOT NULL;

```

执行上面查询语句，结果如下 -

```

+-----+-----+-----+-----+
| id | first_name | last_name | source      | email
| phone |
+-----+-----+-----+-----+
| 1 | John       | Doe        | Web Search | john.doe@yibai.com
| NULL |
+-----+-----+-----+-----+
| 1 row in set

```

即使 `NULL` 不等于 `NULL`，`GROUP BY` 子句中视两个 `NULL` 值相等。

```
SELECT
    email, count(*)
FROM
    leads
GROUP BY email;
```

该查询只返回两行，因为其邮箱(`email`)列为 `NULL` 的行被分组为一行，结果如下所示 -

email	count(*)
<code>NULL</code>	<code>2</code>
<code>john.doe@yiibai.com</code>	<code>1</code>

`2 rows in set`

MySQL NULL 和 UNIQUE 索引

在列上使用唯一约束或 `UNIQUE` 索引时，可以在该列中插入多个 `NULL` 值。这是非常好的，因为在这种情况下，MySQL 认为 `NULL` 值是不同的。

我们通过为 `phone` 列创建一个 `UNIQUE` 紴引来验证这一点。

```
CREATE UNIQUE INDEX idx_phone ON leads(phone);
```

请注意，如果使用 BDB 存储引擎，MySQL 认为 `NULL` 值相等，因此您不能将多个 `NULL` 值插入到具有唯一约束的列中。

MySQL NULL 函数

MySQL 提供了几个有用的功能，很好地处理空值：`IFNULL`，`COALESCE` 和 `NULIF`。

`IFNULL` 函数接受两个参数。如果 `IFNULL` 函数不为 `NULL`，则返回第一个参数，否则返回第二个参数。

例如，如果不是 `NULL`，则以下语句返回电话号码(`phone`)，否则返回 `N/A`，而不是 `NULL`。

```
SELECT
    id, first_name, last_name, IFNULL(phone, 'N/A') phone
FROM
    leads;
```

执行上面查询语句，得到以下结果 -

	<code>id</code>	<code>first_name</code>	<code>last_name</code>	<code>phone</code>
	1	John	Doe	N/A
	2	Lily	Bush	(408) 555-1234
	3	David	William	N/A

3 rows in set

`COALESCE` 函数接受参数列表，并返回第一个非 `NULL` 参数。例如，可以使 `COALESCE` 函数根据信息的优先级按照以下顺序显示线索的联系信息：`phone`，`email` 和 `N/A`。

```
SELECT
    id,
    first_name,
    last_name,
    COALESCE(phone, email, 'N/A') contact
FROM
    leads;
```

执行上面查询语句，得到以下代码 -

id	first_name	last_name	contact
1	John	Doe	john.doe@yiibai.com
2	Lily	Bush	(408)-555-1234
3	David	William	N/A
3 rows in set			

`NULLIF` 函数接受两个参数。如果两个参数相等，则 `NULLIF` 函数返回 `NULL`。否则，它返回第一个参数。

在列中同时具有 `NULL` 和空字符串值时，`NULLIF` 函数很有用。例如，错误地，您将以下行插入到 `leads` 表中：

```
INSERT INTO leads(first_name, last_name, source, email, phone)
VALUE('Thierry', 'Henry', 'Web Search', 'thierry.henry@yiibai.com',
'');
```

`phone` 是一个空字符串: ''，而不是 `NULL`。

如果您想获得潜在客户的联系信息，则最终得到空 `phone`，而不是电子邮件，如下所示：

```
SELECT
    id,
    first_name,
    last_name,
    COALESCE(phone, email, 'N/A') contact
FROM
    leads;
```

执行上面查询语句，得到以下代码 -

<code>id</code>	<code>first_name</code>	<code>last_name</code>	<code>contact</code>
1	John	Doe	<code>john.doe@yiibai.com</code>
2	Lily	Bush	(408)-555-1234
3	David	William	N/A
4	Thierry	Henry	

要解决这个问题，您可以使用 `NULLIF` 函数将电话与空字符串('') 进行比较，如果相等，则返回 `NULL`，否则返回电话号码。

```
SELECT
    id,
    first_name,
    last_name,
    COALESCE(NULLIF(phone, ''), email, 'N/A') contact
FROM
    leads;
```

执行上面查询语句，得到以下代码 -

<code>id</code>	<code>first_name</code>	<code>last_name</code>	<code>contact</code>
1	John	Doe	<code>john.doe@yiibai.com</code>
2	Lily	Bush	(408)-555-1234
3	David	William	N/A
4	Thierry	Henry	<code>thierry.henry@yiibai.com</code>

4 rows in set

在本教程中，您已经学习了如何使用MySQL `NULL` 值，以及如何使用一些方便的函数来处理查询中的 `NULL`。

在本教程中，您将通过使用内置的日期函数来了解如何查询获取MySQL今天的日期数据。

使用内置日期函数获取MySQL今天的日期

有时，您可能希望从表中查询数据，以获取日期列为今天的日期，例如：

```
SELECT
    column_list
FROM
    table_name
WHERE
    expired_date = today;
```

要获取今天的日期，您可以使用 CURDATE() 函数，如下所示：

```
mysql> SELECT CURDATE() today;
+-----+
| today |
+-----+
| 2018-07-31 |
+-----+
1 row in set
```

或者可以从NOW()函数返回的当前时间获取日期部分：

```
mysql> SELECT DATE(NOW()) today;
+-----+
| today |
+-----+
| 2018-07-31 |
+-----+
1 row in set
```

所以查询应该改为：

```
SELECT
    column_list
FROM
    table_name
WHERE
    expired_date = CURDATE();
```

如果 `expired_date` 列包含日期和时间部分，则应使用 `DATE()` 函数仅提取日期部分并将其与当前日期进行比较：

```
SELECT
    column_list
FROM
    table_name
WHERE
    DATE(expired_date) = CURDATE();
```

创建MySQL今天存储过程

如果您在查询中使用 `CURDATE()` 函数，并且希望将其替换为 `today()` 函数以使查询更易读，则可以按如下所示创建名为 `today()` 的自己存储的函数：

```
DELIMITER $$  
CREATE FUNCTION today()  
RETURNS DATE  
BEGIN  
RETURN CURDATE();  
END$$  
DELIMITER ;
```

现在，您可以使用创建的 `today()` 函数，如下所示：

```
mysql> SELECT today();  
+-----+  
| today() |  
+-----+  
| 2017-07-31 |  
+-----+  
1 row in set
```

那么怎么样获取明天日期呢？它应该是简单的：

```
mysql> SELECT today() + interval 1 day Tomorrow;  
+-----+  
| Tomorrow |  
+-----+  
| 2017-08-01 |  
+-----+  
1 row in set
```

同样，获取昨天日期也很容易：

```
mysql> SELECT today() - interval 1 day Yesterday;  
+-----+  
| Yesterday |  
+-----+  
| 2017-07-30 |  
+-----+  
1 row in set
```

在本教程中，您已经学会了如何使用内置的日期函数来获取MySQL今天的日期。您还学习了如何使用MySQL中存储的函数来获取今天日期的功能。

在本教程中，您将学习如何将 NULL 值映射到其他有意义的值。

数据库关系模型创建者 - [E.F.Codd博士](#)在关系数据库理论中引入了 NULL 概念。根据Dr.[E.F.Codd](#)的表述， NULL 表示未知值或缺少信息。

MySQL还支持 NULL 表示缺少或不适用信息的概念。

在数据库表中，您可能会存储包含 NULL 值的数据。但是如果以报表的形式向用户呈现数据时，显示 NULL 值是没有意义的。

要使报告更可读和可理解，必须显示 NULL 值作为其他有意义的值，如未知，缺失或不可用(N/A)。可以使用[IF函数](#)做到这一点。

IF 函数的语法如下：

```
IF(exp,exp_result1,exp_result2);
```

如果 exp 计算结果为 TRUE (当 exp <> 0 和 exp <> NULL)时， IF 函数返回 exp_result1 的值，否则返回 exp_result2 的值。

IF 函数的返回值可以是字符串或数字，具体取决于 exp_result1 和 exp_result2 表达式。

让我们练习一些例子以更好的理解。

假设要使用[示例数据库\(yiibaidb\)](#)中的 customers 表来作演示。

以下是 customers 表中包含： customername ， state 和 country 的部分数据：

```
SELECT
    customername, state, country
FROM
    customers
ORDER BY country;
```

执行上面代码，得到以下结果 -

customername	state	country
Australian Collectors, Co.	Victoria	Australia
Annas Decorations, Ltd	NSW	Australia
Souveniers And Things Co.	NSW	Australia
Australian Gift Network, Co	Queensland	Australia
***** 此处省略了一大波数据 *****		
Handji Gifts & Co	NULL	Singapore
Asian Shopping Network, Co	NULL	Singapore
SAR Distributors, Co ica	Pretoria	South Afr
Euro+ Shopping Channel	NULL	Spain
Diecast Collectables	MA	USA

122 rows in set		

从上面的结果集中，您将看到 state 列的值不适用于某些客户。可以使用 IF 函数将 NULL 值显示为 N/A：

```
SELECT
    customername, IF(state IS NULL, 'N/A', state) state, country
FROM
    customers
ORDER BY country;
```

执行上面查询代码，得到以下结果 -

customername	state	country
Australian Collectors, Co.	Victoria	Australia
Annas Decorations, Ltd	NSW	Australia
Souveniers And Things Co.	NSW	Australia
Australian Gift Network, Co	Queensland	Australia
Australian Collectables, Ltd	Victoria	Australia
Salzburg Collectables	N A	Austria
Mini Auto Werke	N A	Austria
Petit Auto	N A	Belgium
Royale Belge	N A	Belgium
***** 此处省略了一大波数据 *****		
Motor Mint Distributors Inc.	PA	USA
Signal Collectibles Ltd.	CA	USA
Diecast Collectables	MA	USA

122 rows in set

除了 IF 函数外，MySQL提供了IFNULL函数，可以直接处理 NULL 值。以下是 IFNULL 函数的语法：

```
IFNULL(exp,exp_result);
```

重写上代码查询 -

```
SELECT customername,  
       IFNULL(state,"N/A") state,  
       country  
  FROM customers  
 ORDER BY country;
```

如果 `exp` 计算结果为 `NULL` 值，则 `IFNULL` 函数返回 `exp_result` 表达式的值，否则返回 `exp` 表达式的值。以下查询使用 `IFNULL` 函数将 `NULL` 显示为未知，如下所示：

```
SELECT customername,  
       IFNULL(state,"N/A") state,  
       country  
  FROM customers  
 ORDER BY country;
```

执行上面查询代码，得到以下结果 -

customername	state	country
Australian Collectors, Co.	Victoria	Australia
Annas Decorations, Ltd	NSW	Australia
Souveniers And Things Co.	NSW	Australia
Australian Gift Network, Co	Queensland	Australia
Australian Collectables, Ltd	Victoria	Australia
Salzburg Collectables	N/A	Austria
Mini Auto Werke	N/A	Austria
Petit Auto	N/A	Belgium
Royale Belge	N/A	Belgium
***** 此处省略了一大波数据 *****		
Motor Mint Distributors Inc.	PA	USA
Signal Collectibles Ltd.	CA	USA
Diecast Collectables	MA	USA

122 rows in set

在本教程中，您已经学习了如何使用 `IF` 和 `IFNULL` 函数将 `NULL` 值映射到其他更有意义的值，以便以可读方式呈现数据。

在本教程中，您将学习如何使用MySQL注释来记录MySQL中的SQL语句或代码块。

注释

注释可用于记录SQL语句的目的或存储过程中代码块的逻辑。解析SQL代码时，MySQL会忽略注释部分。它只执行除了可执行注释之外的SQL部分，我们将在下一节讨论。

MySQL支持三种注释样式(方式)：

样式一：从 `--` 到行尾

双重冲突注释样式至少需要在第二个破折号之后的空格或控制字符(空格，制表符，换行符等)。

```
SELECT * FROM users; -- This is a comment
```

请注意，标准SQL在第二个破折号后不需要空格。MySQL使用空白来避免某些SQL构造的问题，如：

```
SELECT 10--1;
```

该语句返回 `11`。

样式二：从 `--` 到行尾

参考如下查询语句-

```
SELECT
    lastName, firstName
FROM
    employees
WHERE
    reportsTo = 1002; # get subordinates of Diane
```

样式三：

C语言风格的注释 `/* */` 可以跨越多行。您使用此注释样式记录一个SQL代码块。

```
/*
Get sales rep employees
that reports to Anthony
*/
SELECT
    lastName, firstName
FROM
    employees
WHERE
    reportsTo = 1143
    AND jobTitle = 'Sales Rep';
```

请注意，MySQL不支持嵌套注释。

可执行的注释

MySQL提供可执行注释来支持不同数据库之间的可移植性。这些注释允许嵌入仅能在MySQL中执行，但不能在其他数据库执行SQL代码。

下面说明了可执行注释语法：

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

例如，以下语句使用可执行注释：

```
SELECT 1 /*! +1 */ ;
```

该语句返回 2 而不是1。但是，如果在其他数据库系统中执行，则返回1。

字符串 "#####" 表示可以执行注释的最小版本的MySQL。第一个 # 表示主要版本，例如 5 或 8，第二个 2 个数字(##)是次要版本。最后 2 个(##)表示补丁级别。

例如，以下注释只能在MySQL 5.1.10或更高版本中执行：

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

在本教程中，您已经学会了如何使用MySQL注释来记录MySQL中的SQL代码。

第三章 存储过程

在本节中，您将逐步学习如何在MySQL中编写和开发存储过程。首先，我们向您介绍存储过程的概念，并讨论何时使用它。然后，展示如何使用过程代码的基本元素，如创建存储过程的语句，`if-else`，`case`，`loop`，存储过程的参数。

下面每个教程都包含了易于理解的示例和详细的说明。如果您浏览并学习所有教程，您可以开发MySQL中由简单到复杂的存储过程。

1. MySQL存储过程简介 本教程介绍了MySQL存储过程，它们的优缺点。参考：<http://www.yiibai.com/mysql/introduction-to-sql-stored-procedures.html>

2. MySQL存储过程入门 在本教程中，我们将逐步介绍如何使用 `CREATE PROCEDURE` 语句开发第一个MySQL存储过程。另外，还将展示如何从SQL语句调用存储过程。参考：<http://www.yiibai.com/mysql/getting-started-with-mysql-stored-procedures.html>

3. MySQL存储过程变量 了解并学习MySQL存储过程中的变量，如何声明和使用变量。另外还包括变量的范围。参考：<http://www.yiibai.com/mysql/variables-in-stored-procedures.html>

4. MySQL存储过程参数

本教程将向您介绍如何在存储过程中定义参数，并介绍不同的参数模式，包括 `IN`，`OUT` 和 `INOUT`。参考：<http://www.yiibai.com/mysql/stored-procedures-parameters.html>

5. 返回多个值的MySQL存储过程 本教程将向您展示如何开发返回多个值的存储过程。参考：<http://www.yiibai.com/mysql/stored-procedures-return-multiple-values.html>

6. MySQL IF语句 本教程将向您展示如何使用MySQL `IF` 语句根据条件执行一个SQL代码块。参考：<http://www.yiibai.com/mysql/if-statement.html>

7. MySQL CASE语句 将学习如何使用MySQL `CASE` 语句构建复杂条件。并展示如何使用简单和搜索的 `CASE` 语句。参考：<http://www.yiibai.com/mysql/case-statement.html>

8.在IF和CASE语句使用技巧 我们将给出一些使用技巧，以便可以在存储过程中选择 IF 和 CASE 语句。参考：<http://www.yiibai.com/mysql/conditional-control-if-case-statement-stored-procedures.html>

9.存储过程中的MySQL循环 在本教程中，您将学习如何在存储过程中使用各种循环语句，以基于给定的布尔条件重复执行代码块。

参考：<http://www.yiibai.com/mysql/stored-procedures-loop.html>

10.MySQL游标 显示如何在存储过程中使用MySQL游标循环遍历结果集并一次处理每一行。参考：<http://www.yiibai.com/mysql/cursor.html>

11.在MySQL数据库中列出存储过程 在本教程中，我们将向您展示如何列出MySQL数据库中的所有存储过程。我们还提供一个显示存储过程代码的语句。参考：<http://www.yiibai.com/mysql/listing-stored-procedures-in-mysql-database.html>

12.存储过程中的MySQL错误处理 本教程将向您展示如何使用MySQL处理程序来处理在存储过程中遇到的异常或错误。参考：<http://www.yiibai.com/mysql/error-handling-in-stored-procedures.html>

13.使用MySQL SIGNAL/RESIGNAL语句引发错误条件

在本教程中，您将学习如何使用MySQL SIGNAL 和 RESIGNAL 语句来触发存储过程中的错误条件。参考：<http://www.yiibai.com/mysql/signal-resignal.html>

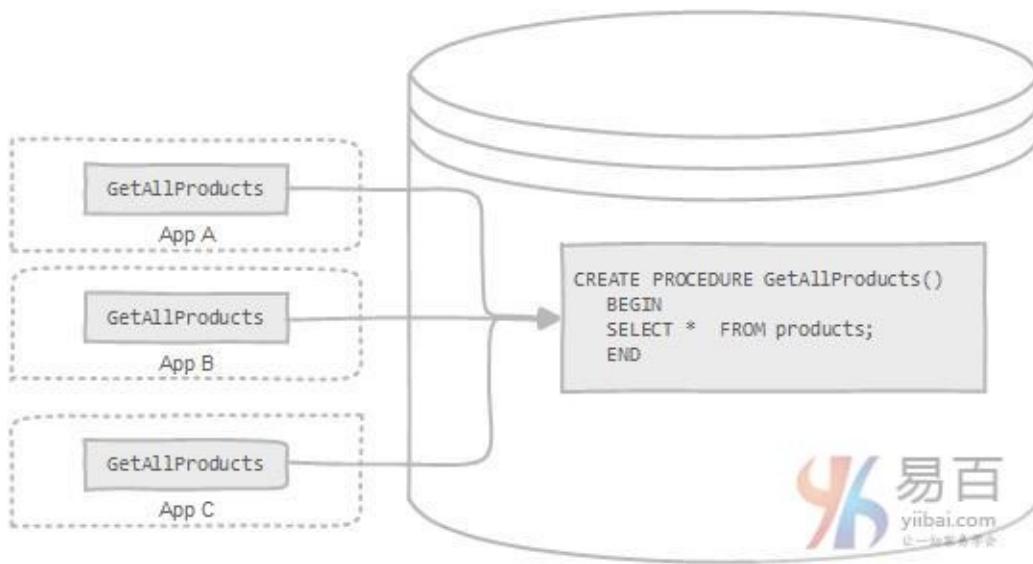
14.MySQL存储函数

在本教程中，您将学习如何使用 CREATE FUNCTION 语句创建MySQL存储函数。参考：<http://www.yiibai.com/mysql/stored-function.html>

在本教程中，您将学习MySQL存储过程什么以及其相关概念，并了解MySQL存储过的优缺点。

存储过程的定义

存储过程是存储在数据库目录中的一段声明性SQL语句。触发器，其他存储过程以及Java，Python，PHP等应用程序可以调用存储过程。



自身的存储过程称为递归存储过程。大多数数据库管理系统支持递归存储过程。但是，MySQL不支持它。在MySQL中实现递归存储过程之前，您应该检查MySQL数据库的版本。

在MySQL中存储过程

MySQL是最受欢迎的开源RDBMS，被社区和企业广泛使用。然而，在它发布的第一个十年期间，它不支持存储过程，[存储函数](#)，[触发器](#)和[事件](#)。自从MySQL 5.0版本以来，这些功能被添加到MySQL数据库引擎，使其更加灵活和强大。

MySQL存储过程的优点

- 通常存储过程有助于提高应用程序的性能。当创建，存储过程被编译之后，就存储在数据库中。但是，MySQL实现的存储过程略有不同。MySQL存储过程按需编译。在编译存储过程之后，MySQL将其放入缓存中。MySQL为每个连接维护自己的存储过程高速缓存。如果应用程序在单个连接中多次使用存储过

程，则使用编译版本，否则存储过程的工作方式类似于查询。

- 存储过程有助于减少应用程序和数据库服务器之间的流量，因为应用程序不必发送多个冗长的SQL语句，而只能发送存储过程的名称和参数。
- 存储的程序对任何应用程序都是可重用的和透明的。存储过程将数据库接口暴露给所有应用程序，以便开发人员不必开发存储过程中已支持的功能。
- 存储的程序是安全的。数据库管理员可以向访问数据库中存储过程的应用程序授予适当的权限，而不向基础数据库表提供任何权限。

除了这些优点之外，存储过程有其自身的缺点，在数据库中使用它们之前，您应该注意这些缺点。

MySQL存储过程的缺点

- 如果使用大量存储过程，那么使用这些存储过程的每个连接的内存使用量将会大大增加。此外，如果您在存储过程中过度使用大量逻辑操作，则CPU使用率也会增加，因为数据库服务器的设计不善于逻辑运算。
- 存储过程的构造使得开发具有复杂业务逻辑的存储过程变得更加困难。
- 很难调试存储过程。只有少数数据库管理系统允许您调试存储过程。不幸的是，MySQL不提供调试存储过程的功能。
- 开发和维护存储过程不容易。开发和维护存储过程通常需要一个不是所有应用程序开发人员拥有的专业技能。这可能会导致应用程序开发和维护阶段的问题。

MySQL存储过程有自己的优点和缺点。开发应用程序时，您应该决定是否应该或不应该根据业务需求使用存储过程。

在下面的教程中，我们将向您展示如何在数据库编程任务中利用MySQL存储过程与许多实际示例。

在本教程中，我们将逐步介绍如何使用 `CREATE PROCEDURE` 语句开发第一个 MySQL 存储过程。另外，我们将向您展示如何从 SQL 语句调用存储过程。

编写第一个MySQL存储过程

我们将开发一个名为 `GetAllProducts()` 的简单存储过程来帮助您熟悉创建存储过程的语法。`GetAllProducts()` 存储过程从 `products` 表中选择所有产品。

启动 `mysql` 客户端工具并键入以下命令：

```
DELIMITER //
CREATE PROCEDURE GetAllProducts()
BEGIN
    SELECT * FROM products;
END //
DELIMITER ;
```

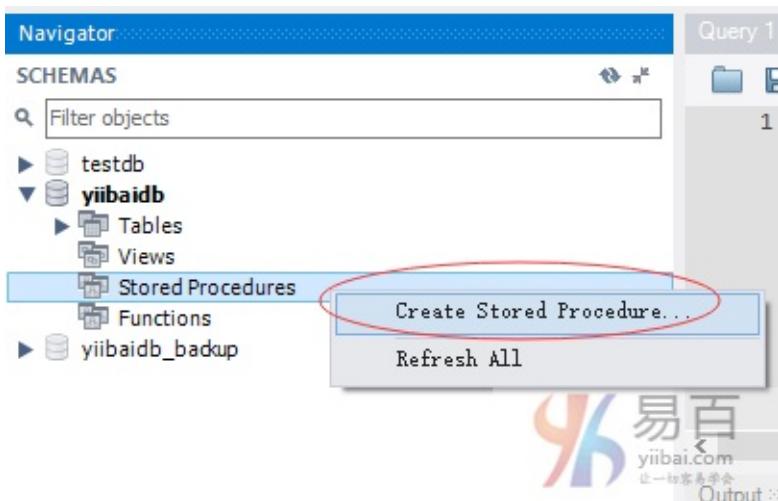
让我们来详细地说明上述存储过程：

- 第一个命令是 `DELIMITER //`，它与存储过程语法无关。`DELIMITER` 语句将标准分隔符 - 分号(;)更改为：//。在这种情况下，分隔符从分号(;)更改为双斜杠 //。为什么我们必须更改分隔符？因为我们想将存储过程作为整体传递给服务器，而不是让 `mysql` 工具一次解释每个语句。在 `END` 关键字之后，使用分隔符 // 来指示存储过程的结束。最后一个命令 (`DELIMITER;`) 将分隔符更改回分号(;)。
- 使用 `CREATE PROCEDURE` 语句创建一个新的存储过程。在 `CREATE PROCEDURE` 语句之后指定存储过程的名称。在这个示例中，存储过程的名称为：`GetAllProducts`，并把括号放在存储过程的名字之后。
- `BEGIN` 和 `END` 之间的部分称为存储过程的主体。将声明性 SQL 语句放在主体中以处理业务逻辑。在这个存储过程中，我们使用一个简单的 `SELECT` 语句来查询 `products` 表中的数据。

在 `mysql` 客户端工具中编写存储过程非常繁琐，特别是当存储过程复杂时。大多数用于 MySQL 的 GUI 工具允许您通过直观的界面创建新的存储过程。

例如，在 `MySQL Workbench` 中，您可以如下创建一个新的存储过程：

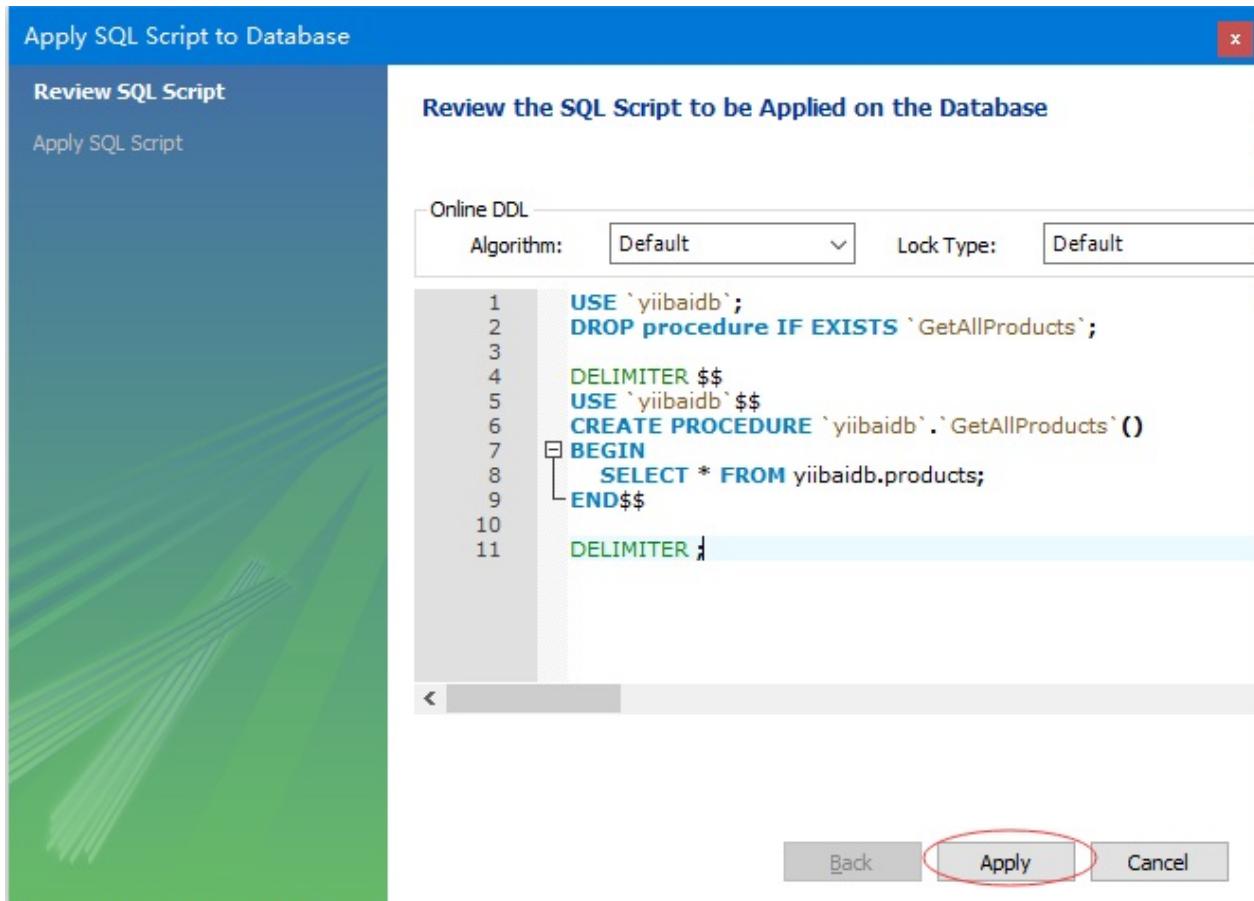
首先，右键单击 `Stored Procedures...` 并选择“`Create Stored Procedure...`”菜单项。



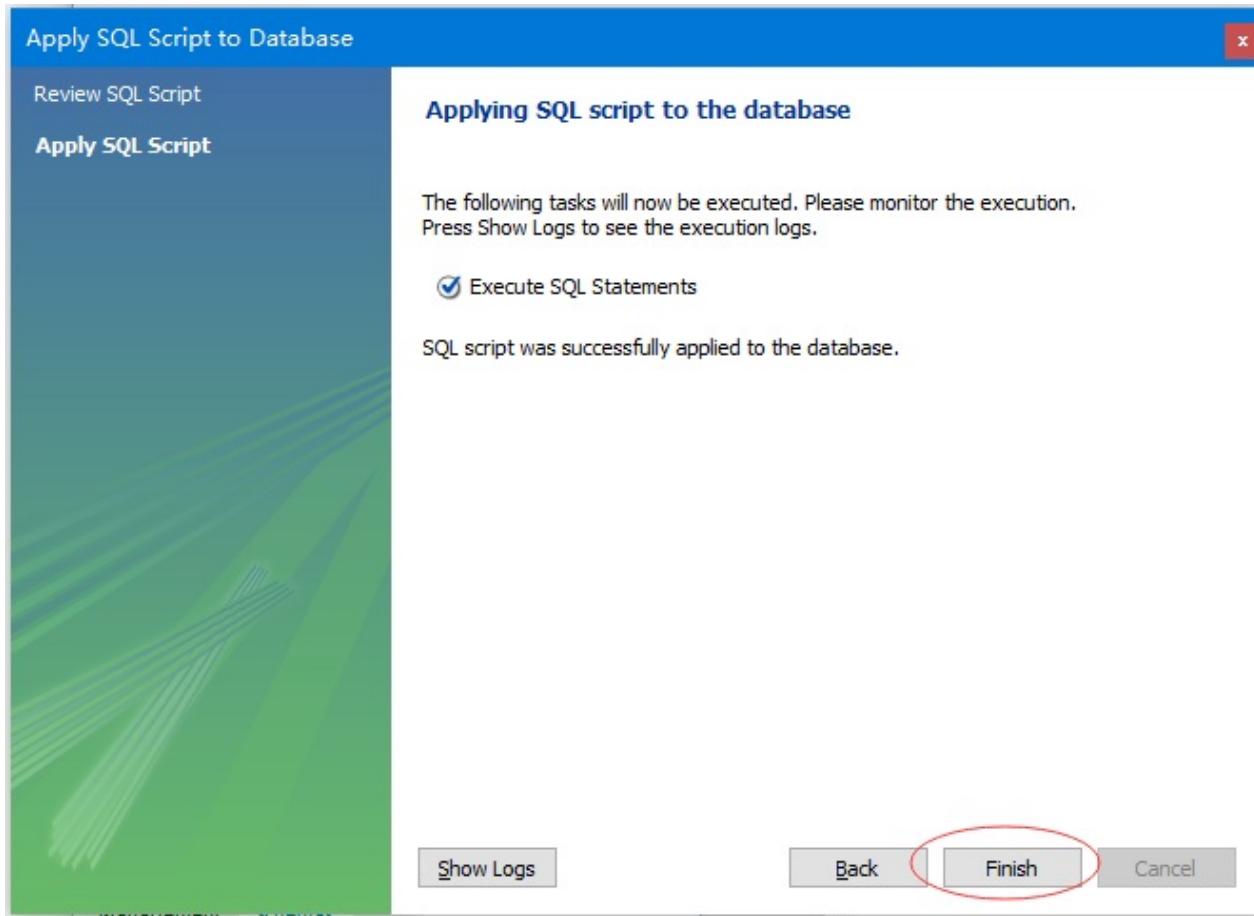
接下来，编写存储过程代码，然后单击*Apply*按钮

```
CREATE PROCEDURE `yiibaidb`.`GetAllProducts`()  
BEGIN  
    SELECT * FROM yiibaidb.products;  
END
```

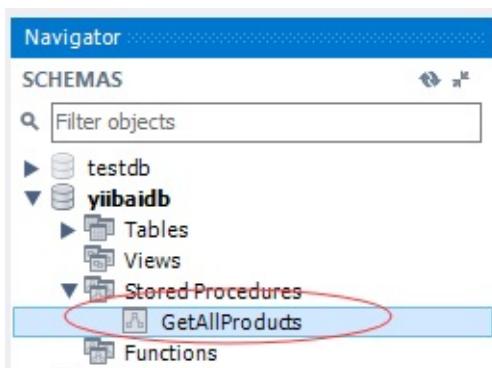
然后，您可以在MySQL将其存储在数据库中之前查看代码。如果一切都是正确的，点击*Apply*按钮。如下所示 -



之后，MySQL将存储过程编译并放入数据库目录中；单击*Fished*按钮完成。



最后，可以在 `yiibaidb` 数据库的例程下看到上面所创建的新存储过程。如下图所示 -



到此，我们已经成功地创建了一个存储过程。下面我们将学习如何使用它。

调用存储过程

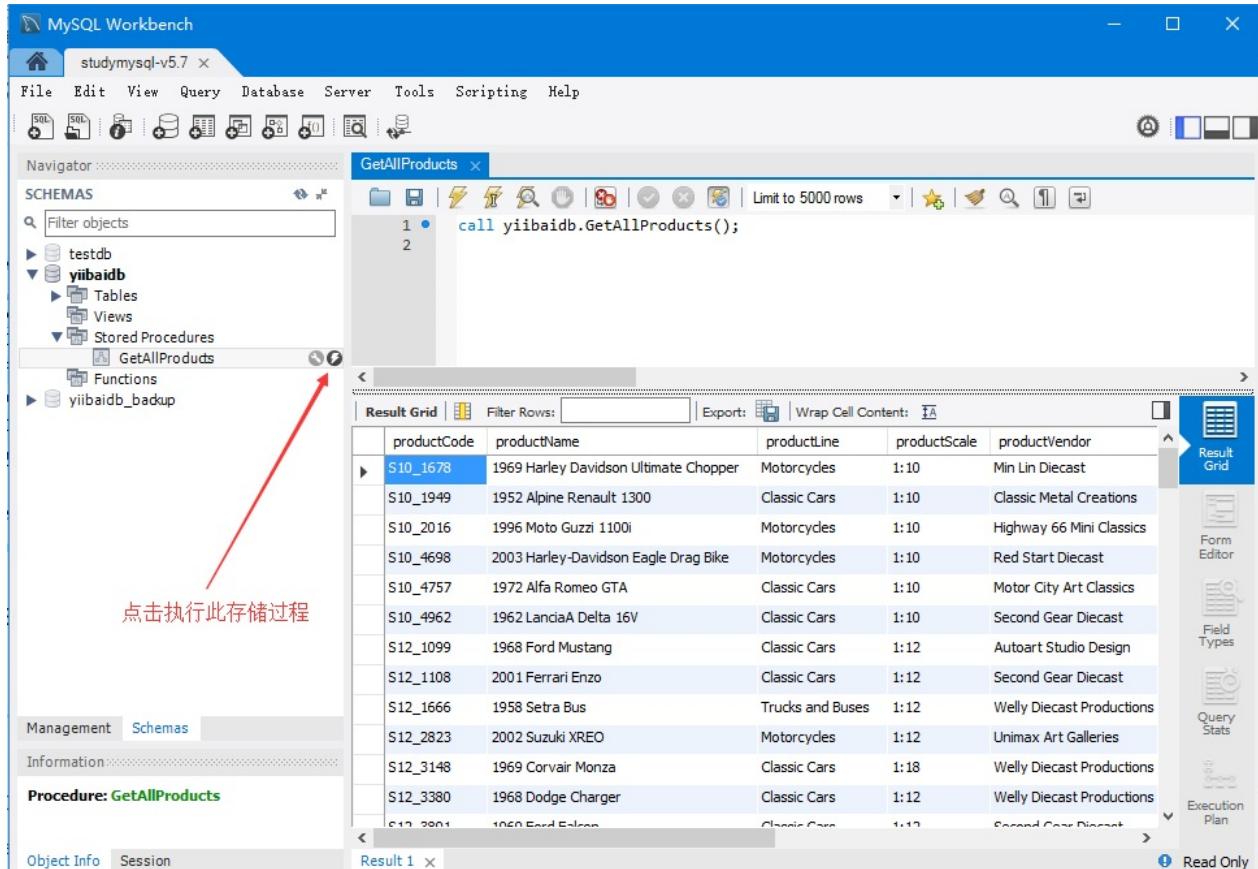
要调用存储过程，可以使用以下SQL命令：

```
CALL STORED_PROCEDURE_NAME();
```

使用 `CALL` 语句调用存储过程，例如调用 `GetAllProducts()` 存储过程，则使用以下语句：

```
CALL GetAllProducts();
```

如果您执行上述语句，将查询获得 `products` 表中的所有产品。如下图所示 -



在本教程中，您已经学习了如何使用 `CREATE PROCEDURE` 语句编写一个简单的存储过程，并使用 `CALL` 语句从SQL语句中调用它。

在本教程中，您将了解和学习存储过程中的变量，包括如何声明和使用变量。此外，您将了解变量的作用域(范围)。

变量是一个命名数据对象，变量的值可以在存储过程执行期间更改。我们通常使用存储过程中的变量来保存直接/间接结果。这些变量是存储过程的本地变量。

注意：变量必须先声明后，才能使用它。

声明变量

要在存储过程中声明一个变量，可以使用 `DECLARE` 语句，如下所示：

```
DECLARE variable_name datatype(size) DEFAULT default_value;
```

下面来更详细地解释上面的语句：

- 首先，在 `DECLARE` 关键字后面要指定变量名。变量名必须遵循MySQL表列名称的命名规则。
- 其次，指定变量的数据类型及其大小。变量可以有任何MySQL数据类型，如 `INT`，`VARCHAR`，`DATETIME` 等。
- 第三，当声明一个变量时，它的初始值为 `NULL`。但是可以使用 `DEFAULT` 关键字为变量分配默认值。

例如，可以声明一个名为 `total_sale` 的变量，数据类型为 `INT`，默认值为 `0`，如下所示：

```
DECLARE total_sale INT DEFAULT 0;
```

MySQL允许您使用单个 `DECLARE` 语句声明共享相同数据类型的两个或多个变量，如下所示：

```
DECLARE x, y INT DEFAULT 0;
```

我们声明了两个整数变量 `x` 和 `y`，并将其默认值设置为 `0`。

分配变量值

当声明了一个变量后，就可以开始使用它了。要为变量分配一个值，可以使用 `SET` 语句，例如：

```
DECLARE total_count INT DEFAULT 0;
SET total_count = 10;
```

上面语句中，分配 `total_count` 变量的值为 `10`。

除了 `SET` 语句之外，还可以使用 `SELECT INTO` 语句将查询的结果分配给一个变量。请参阅以下示例：

```
DECLARE total_products INT DEFAULT 0

SELECT COUNT(*) INTO total_products
FROM products
```

在上面的例子中：

- 首先，声明一个名为 `total_products` 的变量，并将其值初始化为 `0`。
- 然后，使用 `SELECT INTO` 语句来分配值给 `total_products` 变量，从[示例数据库\(yiibaidb\)](#)中的 `products` 表中选择的产品数量。

变量范围(作用域)

一个变量有自己的范围(作用域)，它用来定义它的生命周期。如果在存储过程中声明一个变量，那么当达到存储过程的 `END` 语句时，它将超出范围，因此在其它代码块中无法访问。

如果您在 `BEGIN END` 块内声明一个变量，那么如果达到 `END`，它将超出范围。可以在不同的作用域中声明具有相同名称的两个或多个变量，因为变量仅在自己的作用域中有效。但是，在不同范围内声明具有相同名称的变量不是很好的编程习惯。

以 `@` 符号开头的变量是会话变量。直到会话结束前它可用和可访问。

在本教程中，我们向您展示了如何在存储过程中声明变量，并讨论了变量的范围(作用域)。

在本教程中，您将学习如何编写具有参数的MySQL存储过程。还将通过几个存储过程示例来了解不同类型的参数。

MySQL存储过程参数简介

在现实应用中，开发的存储过程几乎都需要参数。这些参数使存储过程更加灵活和有用。在MySQL中，参数有三种模式： IN ， OUT 或 INOUT 。

- IN - 是默认模式。在存储过程中定义 IN 参数时，调用程序必须将参数传递给存储过程。另外， IN 参数的值被保护。这意味着即使在存储过程中更改了 IN 参数的值，在存储过程结束后仍保留其原始值。换句话说，存储过程只使用 IN 参数的副本。
- OUT - 可以在存储过程中更改 OUT 参数的值，并将其更改后新值传递回调用程序。请注意，存储过程在启动时无法访问 OUT 参数的初始值。
- INOUT - INOUT 参数是 IN 和 OUT 参数的组合。这意味着调用程序可以传递参数，并且存储过程可以修改 INOUT 参数并将新值传递回调用程序。

在存储过程中定义参数的语法如下：

```
MODE param_name param_type(param_size)
```

上面语法说明如下 -

- 根据存储过程中参数的目的， MODE 可以是 IN ， OUT 或 INOUT 。
- param_name 是参数的名称。参数的名称必须遵循MySQL中列名的命名规则。
- 在参数名之后是它的数据类型和大小。和变量一样，参数的数据类型可以是任何有效的MySQL数据类型。

如果存储过程有多个参数，则每个参数由逗号(,)分隔。

让我们练习一些例子来更好的理解。我们将使用示例数据库(yiibaidb)中的表进行演示。

MySQL存储过程参数示例

1.IN参数示例

以下示例说明如何使用 `GetOfficeByCountry` 存储过程中的 `IN` 参数来查询选择位于特定国家/地区的办公室。

```

USE `yibaidb`;
DROP procedure IF EXISTS `GetOfficeByCountry`;

DELIMITER $$
USE `yibaidb`$$
CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))
BEGIN
    SELECT *
    FROM offices
    WHERE country = countryName;
END$$

DELIMITER ;

```

`countryName` 是存储过程的 `IN` 参数。在存储过程中，我们查询位于 `countryName` 参数指定的国家/地区的所有办公室。

假设我们想要查询在美国(`USA`)的所有办事处，我们只需要将一个值(`USA`)传递给存储过程，如下所示：

```
CALL GetOfficeByCountry('USA');
```

执行上面查询语句，得到以下结果 -

The screenshot shows the MySQL Workbench interface with a query editor titled "GetOfficeByCountry*". The query is:

```
CALL GetOfficeByCountry('USA');
```

The results grid displays the following data:

	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode
▶	1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080
	2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107
	3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022

要在法国获得所有办事处，我们将 France 字符串传递给 GetOfficeByCountry 存储过程，如下所示：

```
CALL GetOfficeByCountry('France')
```

2.OUT参数示例

以下存储过程通过订单状态返回订单数量。它有两个参数：

- orderStatus : IN 参数，它是要对订单计数的订单状态。
- total : 存储指定订单状态的订单数量的 OUT 参数。

以下是 CountOrderByStatus 存储过程的源代码。

```
USE `yibaidb`;
DROP procedure IF EXISTS `CountOrderByStatus`;

DELIMITER $$

CREATE PROCEDURE CountOrderByStatus(
    IN orderStatus VARCHAR(25),
    OUT total INT)
BEGIN
    SELECT count(orderNumber)
    INTO total
    FROM orders
    WHERE status = orderStatus;
END$$
DELIMITER ;
```

要获取发货订单的数量，我们调用 CountOrderByStatus 存储过程，并将订单状态传递为已发货，并传递参数(@total)以获取返回值。

```
CALL CountOrderByStatus('Shipped',@total);
SELECT @total;
```

执行上面查询语句后，得到以下结果 -

```
+-----+
| @total |
+-----+
| 303 |
+-----+
1 row in set
```

要获取正在处理的订单数量，调用 `CountOrderByStatus` 存储过程，如下所示：

执行上面查询语句后，得到以下结果 -

```
+-----+
| total_in_process |
+-----+
| 7 |
+-----+
1 row in set
```

INOUT 参数示例

以下示例演示如何在存储过程中使用 `INOUT` 参数。如下查询语句 -

```
DELIMITER $$  
CREATE PROCEDURE set_counter(INOUT count INT(4), IN inc INT(4))  
BEGIN  
    SET count = count + inc;  
END$$  
DELIMITER ;
```

上面查询语句是如何运行的？

- `set_counter` 存储过程接受一个 `INOUT` 参数(`count`)和一个 `IN` 参数(`inc`)。
- 在存储过程中，通过 `inc` 参数的值增加计数器(`count`)。

下面来看看如何调用 `set_counter` 存储过程：

```
SET @counter = 1;
CALL set_counter(@counter,1); -- 2
CALL set_counter(@counter,1); -- 3
CALL set_counter(@counter,5); -- 8
SELECT @counter; -- 8
```

在本教程中，我们向您展示了如何在存储过程中定义参数，并介绍了不同的参数模式：`IN`，`OUT` 和 `INOUT`。

在本教程中，您将学习如何编写/开发返回多个值的存储过程。

MySQL 存储函数 只返回一个值。要开发返回多个值的 存储过程，需要使用带有 `INOUT` 或 `OUT` 参数的存储过程。

如果您不熟悉 `INPUT` 或 `OUT` 参数的用法，请查看[存储过程参数教程](#)的详细信息。

返回多个值的存储过程示例

我们来看看示例数据库(yiibaidb)中的 `orders` 表。

```
mysql> desc orders;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| orderNumber | int(11) | NO | PRI | NULL |
| orderDate | date | NO | | NULL |
| requiredDate | date | NO | | NULL |
| shippedDate | date | YES | | NULL |
| status | varchar(15) | NO | | NULL |
| comments | text | YES | | NULL |
| customerNumber | int(11) | NO | MUL | NULL |
+-----+-----+-----+-----+-----+
7 rows in set
```

以下存储过程接受客户编号，并返回发货(`shipped`)，取消(`canceled`)，解决(`resolved`)和争议(`disputed`)的订单总数。

```
DELIMITER $$

CREATE PROCEDURE get_order_by_cust(
    IN cust_no INT,
    OUT shipped INT,
    OUT canceled INT,
    OUT resolved INT,
    OUT disputed INT)
BEGIN
    -- shipped
```

```

SELECT
    count(*) INTO shipped
FROM
    orders
WHERE
    customerNumber = cust_no
        AND status = 'Shipped';

-- canceled
SELECT
    count(*) INTO canceled
FROM
    orders
WHERE
    customerNumber = cust_no
        AND status = 'Canceled';

-- resolved
SELECT
    count(*) INTO resolved
FROM
    orders
WHERE
    customerNumber = cust_no
        AND status = 'Resolved';

-- disputed
SELECT
    count(*) INTO disputed
FROM
    orders
WHERE
    customerNumber = cust_no
        AND status = 'Disputed';

END

```

除 IN 参数之外，存储过程还需要 4 个额外的 OUT 参数：shipped，canceled，resolved 和 disputed。在存储过程中，使用带有 COUNT 函数的 SELECT 语句根据订单状态获取相应的订单总数，并将其分配给相应的参数。

要使用 `get_order_by_cust` 存储过程，可以传递客户编号和四个用户定义的变量来获取输出值。

执行存储过程后，使用 `SELECT` 语句输出变量值。

```
+-----+-----+-----+-----+
| @shipped | @canceled | @resolved | @disputed |
+-----+-----+-----+-----+
|      22 |         0 |         1 |         1 |
+-----+-----+-----+-----+
1 row in set
```

从**PHP**调用返回多个值的存储过程

以下代码片段显示如何从**PHP**程序中调用返回多个值的存储过程。

```

<?php
/**
 * Call stored procedure that return multiple values
 * @param $customerNumber
 */
function call_sp($customerNumber)
{
    try {
        $pdo = new PDO("mysql:host=localhost;dbname=yiibaidb", 'root', '123456');

        // execute the stored procedure
        $sql = 'CALL get_order_by_cust(:no,@shipped,@canceled,@resolved,@disputed)';
        $stmt = $pdo->prepare($sql);

        $stmt->bindParam(':no', $customerNumber, PDO::PARAM_INT);

        $stmt->execute();
        $stmt->closeCursor();

        // execute the second query to get values from OUT parameter
        $r = $pdo->query("SELECT @shipped,@canceled,@resolved,@disputed")
            ->fetch(PDO::FETCH_ASSOC);
        if ($r) {
            printf('Shipped: %d, Canceled: %d, Resolved: %d, Disputed: %d',
                $r['@shipped'],
                $r['@canceled'],
                $r['@resolved'],
                $r['@disputed']);
        }
    } catch (PDOException $pe) {
        die("Error occurred:" . $pe->getMessage());
    }
}

call_sp(141);

```

在 @ 符号之前的用户定义的变量与数据库连接相关联，因此它们可用于在调用之间进行访问。

在本教程中，我们向您展示了如何编写/开发返回多个值的存储过程以及如何从 PHP 调用它。

在本教程中，您将学习如何使用MySQL IF语句来根据条件执行一个SQL代码块。

MySQL IF语句允许您根据表达式的某个条件或值结果来执行一组SQL语句。要在MySQL中形成一个表达式，可以结合文字，[变量](#)，运算符，甚至函数来组合。表达式可以返回 TRUE，FALSE 或 NULL，这三个值之一。

请注意，有一个[IF函数](#)与本教程中指定的 IF语句是不同的。

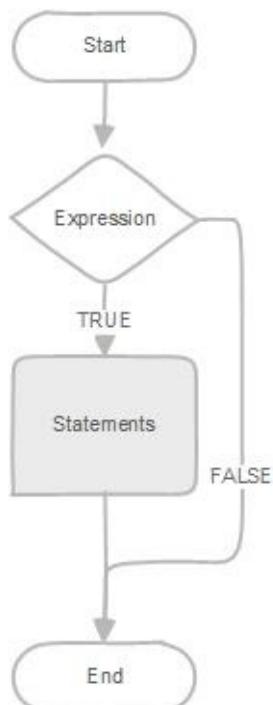
MySQL IF语句语法

下面说明了 IF语句的语法：

```
IF expression THEN  
    statements;  
END IF;
```

如果表达式(expression)计算结果为 TRUE，那么将执行 statements语句，否则控制流将传递到 END IF之后的下一个语句。

以下流程图演示了 IF语句的执行过程：

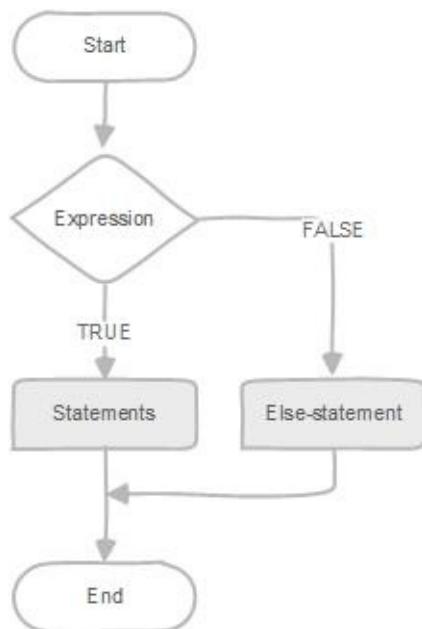


MySQL IF ELSE语句

如果表达式计算结果为 FALSE 时执行语句，请使用 IF ELSE 语句，如下所示：

```
IF expression THEN  
    statements;  
ELSE  
    else statements;  
END IF;
```

以下流程图说明了 IF ELSE 语句的执行过程：

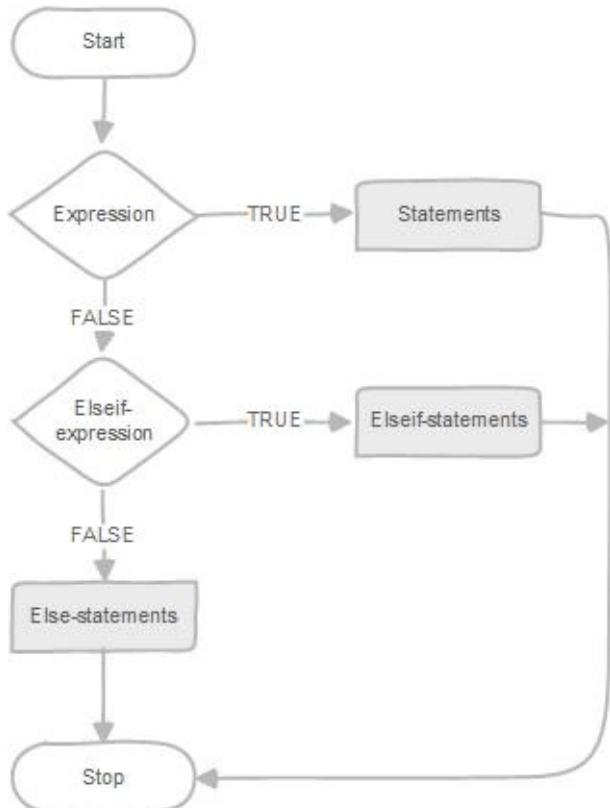


MySQL IF ELSEIF ELSE语句

如果要基于多个表达式有条件地执行语句，则使用 IF ELSEIF ELSE 语句如下：

```
IF expression THEN  
    statements;  
ELSEIF elseif expression THEN  
    elseif statements;  
...  
ELSE  
    else statements;  
END IF;
```

如果表达式(`expression`)求值为 `TRUE` , 则 `IF` 分支中的语句(`statements`)将执行; 如果表达式求值为 `FALSE` , 则如果 `elseif_expression` 的计算结果为 `TRUE` , MySQL将执行 `elseif-expression` , 否则执行 `ELSE` 分支中的 `else-statements` 语句。具体流程如下 -



MySQL IF语句示例

以下示例说明如何使用 `IF ESLEIF ELSE` 语句, `GetCustomerLevel()` 存储过程接受客户编号和客户级别的两个参数。

首先, 它从 `customers` 表中获得信用额度

然后, 根据信用额度, 它决定客户级别: `PLATINUM` , `GOLD` 和 `SILVER` 。

参数 `p_customerlevel` 存储客户的级别, 并由调用程序使用。

```
USE yiibaidb;

DELIMITER $$

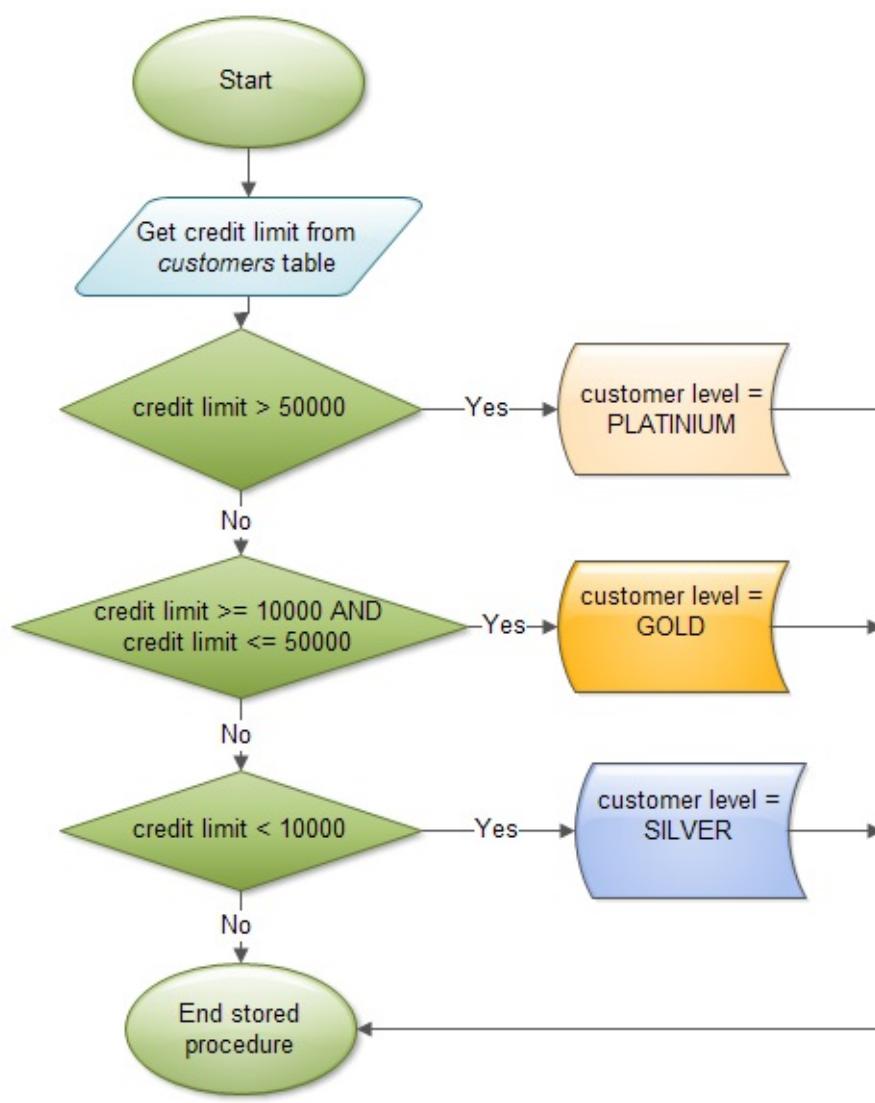
CREATE PROCEDURE GetCustomerLevel(
    in p_customerNumber int(11),
    out p_customerLevel varchar(10))
BEGIN
    DECLARE creditlim double;

    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber = p_customerNumber;

    IF creditlim > 50000 THEN
        SET p_customerLevel = 'PLATINUM';
    ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
        SET p_customerLevel = 'GOLD';
    ELSEIF creditlim < 10000 THEN
        SET p_customerLevel = 'SILVER';
    END IF;

END$$
```

以下流程图演示了确定客户级别的逻辑 -



在本教程中，您已经学会了如何使用MySQL IF 语句根据条件执行一个SQL代码块。

在本教程中，您将学习如何使用MySQL CASE 语句在存储的程序中构造复杂的条件语句。

除了IF语句，MySQL提供了一个替代的条件语句 CASE 。 MySQL CASE 语句使代码更加可读和高效。

CASE 语句有两种形式：简单的搜索 CASE 语句。

简单CASE语句

我们来看一下简单 CASE 语句的语法：

```
CASE case_expression
    WHEN when_expression_1 THEN commands
    WHEN when_expression_2 THEN commands
    ...
    ELSE commands
END CASE;
```

您可以使用简单 CASE 语句来检查表达式的值与一组唯一值的匹配。

case_expression 可以是任何有效的表达式。我们将 case_expression 的值与每个 WHEN 子句中的 when_expression 进行比较，例如 when_expression_1 ， when_expression_2 等。如果 case_expression 和 when_expression_n 的值相等，则执行相应的 WHEN 分支中的命令(commands)。

如果 WHEN 子句中的 when_expression 与 case_expression 的值匹配，则 ELSE 子句中的命令将被执行。 ELSE 子句是可选的。如果省略 ELSE 子句，并且找不到匹配项，MySQL将引发错误。

以下示例说明如何使用简单的 CASE 语句：

```

DELIMITER $$

CREATE PROCEDURE GetCustomerShipping(
    in p_customerNumber int(11),
    out p_shipping      varchar(50))
BEGIN
    DECLARE customerCountry varchar(50);

    SELECT country INTO customerCountry
    FROM customers
    WHERE customerNumber = p_customerNumber;

    CASE customerCountry
    WHEN 'USA' THEN
        SET p_shipping = '2-day Shipping';
    WHEN 'Canada' THEN
        SET p_shipping = '3-day Shipping';
    ELSE
        SET p_shipping = '5-day Shipping';
    END CASE;

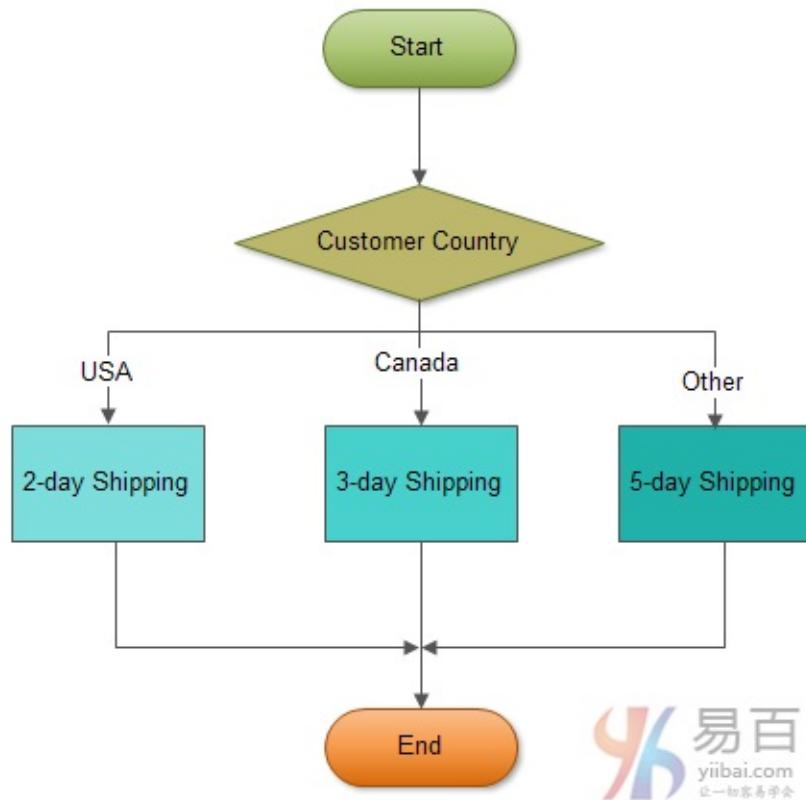
END$$

```

上面存储过程是如何工作的？

- `GetCustomerShipping` 存储过程接受客户编号作为 `IN` 参数，并根据客户所在国家返回运送时间。
- 在存储过程中，首先，我们根据输入的客户编号得到客户的国家。然后使用简单 `CASE` 语句来比较客户的国家来确定运送期。如果客户位于美国(`USA`)，则运送期为 2 天。如果客户在加拿大，运送期为 3 天。来自其他国家的客户则需要 5 天的运输时间。

以下流程图显示了确定运输时间的逻辑。



以下是上述存储过程的测试脚本：

```

SET @customerNo = 112;

SELECT country into @country
FROM customers
WHERE customernumber = @customerNo;

CALL GetCustomerShipping(@customerNo,@shipping);

SELECT @customerNo AS Customer,
       @country   AS Country,
       @shipping   AS Shipping;
  
```

执行上面代码，得到以下结果 -

```
+-----+-----+-----+
| Customer | Country | Shipping |
+-----+-----+-----+
| 112     | USA      | 2-day Shipping |
+-----+-----+-----+
1 row in set
```

可搜索CASE语句

简单 CASE 语句仅允许您将表达式的值与一组不同的值进行匹配。为了执行更复杂的匹配，如范围，您可以使用可搜索 CASE 语句。可搜索 CASE 语句等同于 IF 语句，但是它的构造更加可读。

以下说明可搜索 CASE 语句的语法：

```
CASE
WHEN condition_1 THEN commands
WHEN condition_2 THEN commands
...
ELSE commands
END CASE;
```

MySQL评估求值 WHEN 子句中的每个条件，直到找到一个值为 TRUE 的条件，然后执行 THEN 子句中的相应命令(commands)。

如果没有一个条件为 TRUE，则执行 ELSE 子句中的命令(commands)。如果不指定 ELSE 子句，并且没有一个条件为 TRUE，MySQL将发出错误消息。

MySQL不允许在 THEN 或 ELSE 子句中使用空的命令。如果您不想处理 ELSE 子句中的逻辑，同时又要防止MySQL引发错误，则可以在 ELSE 子句中放置一个空的 BEGIN END 块。

以下示例演示如何使用可搜索 CASE 语句来根据客户的信用额度来查找客户级： SILVER， GOLD 或 PLATINUM 。

```

DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
    in p_customerNumber int(11),
    out p_customerLevel varchar(10))
BEGIN
    DECLARE creditlim double;

    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber = p_customerNumber;

    CASE
        WHEN creditlim > 50000 THEN
            SET p_customerLevel = 'PLATINUM';
        WHEN (creditlim <= 50000 AND creditlim >= 10000) THEN
            SET p_customerLevel = 'GOLD';
        WHEN creditlim < 10000 THEN
            SET p_customerLevel = 'SILVER';
    END CASE;

END$$

```

在上面查询语句逻辑中，如果信用额度是：

- 大于 50K，则客户是 PLATINUM 客户。
- 小于 50K，大于 10K，则客户是 GOLD 客户。
- 小于 10K，那么客户就是 SILVER 客户。

我们可以通过执行以下测试脚本来测试存储过程：

```

CALL GetCustomerLevel(112,@level);
SELECT @level AS 'Customer Level';

```

执行上面查询语句，得到以下结果 -

Customer Level
PLATINUM

1 row in set

在本教程中，我们向您展示了如何使用两种形式的MySQL CASE 语句，包括简单 CASE 语句和可搜索 CASE 语句。

在本教程中，我们将给您一些技巧，以便您在存储过程中在什么情况分别选择 IF 和 CASE 语句。

MySQL提供 IF 和 CASE 语句，使您能够根据某些条件(称为流控制)执行一个SQL代码块。那么您应该使用什么语句？对于大多数开发人员，在 IF 和 CASE 之间进行选择只是个人偏好的问题。但是，当您决定使用 IF 或 CASE 时，应该考虑以下几点：

- 当将单个表达式与唯一值的范围进行比较时，简单CASE语句比IF语句更易读。另外，简单 CASE 语句比 IF 语句更有效率。
- 当您根据多个值检查复杂表达式时， IF 语句更容易理解。
- 如果您选择使用 CASE 语句，则必须确保至少有一个 CASE 条件匹配。否则，需要定义一个[错误处理](#)程序来捕获错误。 IF 语句则不需要处理错误。
- 在大多数组织(公司)中，总是有一些所谓的开发指导文件，为开发人员提供了编程风格的命名约定和指导，那么您应参考本文档并遵循开发实践。
- 在某些情况下， IF 和 CASE 混合使用反而使您的存储过程更加可读和高效。

在本教程中，将学习如何使用各种MySQL循环语句(包括 `WHILE` , `REPEAT` 和 `LOOP`)来根据条件反复运行代码块。

MySQL提供循环语句，允许您根据条件重复执行一个SQL代码块。MySQL中有三个循环语句：`WHILE` , `REPEAT` 和 `LOOP` 。

我们将在以下部分中更详细地检查每个循环语句。

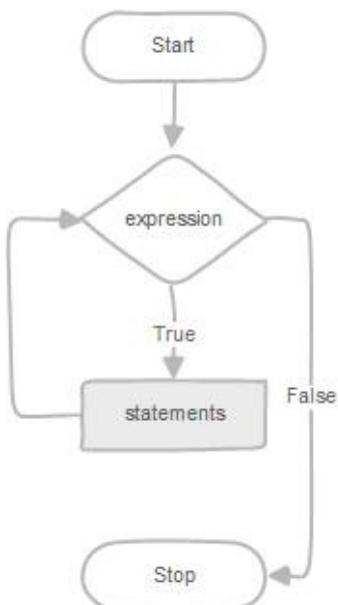
WHILE 循环

`WHILE` 语句的语法如下：

```
WHILE expression DO
    statements
END WHILE
```

`WHILE` 循环在每次迭代开始时检查表达式。如果 `expression` evaluates 为 `TRUE` ，MySQL将执行 `WHILE` 和 `END WHILE` 之间的语句，直到 `expression` evaluates 为 `FALSE` 。 `WHILE` 循环称为预先测试条件循环，因为它总是在执行前检查语句的表达式。

下面的流程图说明了 `WHILE` 循环语句：



以下是在存储过程中使用 `WHILE` 循环语句的示例：

```

DELIMITER $$

DROP PROCEDURE IF EXISTS test_mysql_while_loop$$
CREATE PROCEDURE test_mysql_while_loop()
BEGIN
DECLARE x INT;
DECLARE str VARCHAR(255);

SET x = 1;
SET str = '';

WHILE x <= 5 DO
SET str = CONCAT(str,x,' ');
SET x = x + 1;
END WHILE;

SELECT str;
END$$
DELIMITER ;

```

在上面的 `test_mysql_while_loop` 存储过程中：

- 首先，重复构建 `str` 字符串，直到 `x` 变量的值大于 `5`。
- 然后，使用 `SELECT` 语句 显示最终的字符串。

要注意，如果不初始化 `x` 变量的值，那么它默认值为 `NULL`。因此，`WHILE` 循环语句中的条件始终为 `TRUE`，并且您将有一个不确定的循环，这是不可预料的。

下面来测试 `test_mysql_while_looppstored` 调用存储过程：

```
CALL test_mysql_while_loop();
```

执行上面查询语句，得到以下结果 -

```
mysql> CALL test_mysql_while_loop();
+-----+
| str |
+-----+
| 1,2,3,4,5, |
+-----+
1 row in set

Query OK, 0 rows affected
```

REPEAT循环

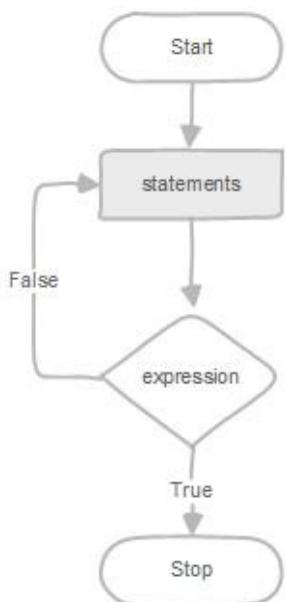
REPEAT 循环语句的语法如下：

```
REPEAT
  statements;
UNTIL expression
END REPEAT
```

首先，MySQL执行语句，然后评估求值表达式(expression)。如果表达式(expression)的计算结果为 FALSE ，则MySQL将重复执行该语句，直到该表达式计算结果为 TRUE 。

因为 REPEAT 循环语句在执行语句后检查表达式(expression)，因此 REPEAT 循环语句也称为测试后循环。

下面的流程图说明了 REPEAT 循环语句的执行过程：



我们可以使用 REPEAT 循环语句重写 test_mysql_while_loop 存储过程，使用 WHILE 循环语句：

```

DELIMITER $$

DROP PROCEDURE IF EXISTS mysql_test_repeat_loop$$
CREATE PROCEDURE mysql_test_repeat_loop()
BEGIN
DECLARE x INT;
DECLARE str VARCHAR(255);

SET x = 1;
      SET str = '';

REPEAT
SET str = CONCAT(str,x,',');
SET x = x + 1;
      UNTIL x > 5
      END REPEAT;

      SELECT str;
END$$
DELIMITER ;
  
```

要注意的是 UNTIL 表达式中没有分号(;)。

执行上面查询语句，得到以下结果 -

```
mysql> CALL mysql_test_repeat_loop();
+-----+
| str |
+-----+
| 1,2,3,4,5, |
+-----+
1 row in set

Query OK, 0 rows affected
```

LOOP，LEAVE和ITERATE语句

有两个语句允许您用于控制循环：

- LEAVE 语句用于立即退出循环，而无需等待检查条件。 LEAVE 语句的工作原理就类似 PHP ， C/C++ ， Java 等其他语言的 break 语句一样。
- ITERATE 语句允许您跳过剩下的整个代码并开始新的迭代。 ITERATE 语句类似于 PHP ， C/C++ ， Java 等中的 continue 语句。

MySQL还有一个 LOOP 语句，它可以反复执行一个代码块，另外还有一个使用循环标签的灵活性。

以下是使用 LOOP 循环语句的示例。

```

CREATE PROCEDURE test_mysql_loop()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);

    SET x = 1;
    SET str = '';

loop_label: LOOP
    IF x > 10 THEN
        LEAVE loop_label;
    END IF;

    SET x = x + 1;
    IF (x mod 2) THEN
        ITERATE loop_label;
    ELSE
        SET str = CONCAT(str, x, ',');
    END IF;
END LOOP;
SELECT str;
END;

```

- 以上存储过程仅构造具有偶数字符串的字符串，例如 2, 4, 6 等。
- 在 LOOP 语句之前放置一个 loop_label 循环标签。
- 如果 x 的值大于 10，则由于 LEAVE 语句，循环被终止。
- 如果 x 的值是一个奇数，ITERATE 语句忽略它下面的所有内容，并开始一个新的迭代。
- 如果 x 的值是偶数，则 ELSE 语句中的块将使用偶数构建字符串。

执行上面查询语句，得到以下结果 -

```
mysql> CALL test_mysql_loop();
+-----+
| str   |
+-----+
| 2,4,6,8,10, |
+-----+
1 row in set

Query OK, 0 rows affected
```

在本教程中，您学习了基于条件重复执行代码块的各种MySQL循环语句。

在本教程中，您将学习如何在存储过程中使用MySQL游标来遍历 `SELECT` 语句返回的结果集。

MySQL游标简介

要处理[存储过程](#)中的结果集，请使用游标。游标允许您迭代查询返回的一组行，并相应地处理每行。

MySQL游标为只读，不可滚动和敏感。

- 只读：无法通过光标更新基础表中的数据。
- 不可滚动：只能按照[SELECT](#)语句确定的顺序获取行。不能以相反的顺序获取行。此外，不能跳过行或跳转到结果集中的特定行。
- 敏感：有两种游标：敏感游标和不敏感游标。敏感游标指向实际数据，不敏感游标使用数据的临时副本。敏感游标比一个不敏感的游标执行得更快，因为它不需要临时拷贝数据。但是，对其他连接的数据所做的任何更改都将影响由敏感游标使用的数据，因此，如果不更新敏感游标所使用的数据，则更安全。
MySQL游标是敏感的。

您可以在[存储过程](#)，[存储函数](#)和[触发器](#)中使用MySQL游标。

使用MySQL游标

首先，必须使用 `DECLARE` 语句声明游标：

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

游标声明必须在[变量](#)声明之后。如果在变量声明之前声明游标，MySQL将会发出一个错误。游标必须始终与 `SELECT` 语句相关联。

接下来，使用 `OPEN` 语句打开游标。`OPEN` 语句初始化游标的結果集，因此您必须在从結果集中提取行之前调用 `OPEN` 语句。

```
OPEN cursor_name;
```

然后，使用 `FETCH` 语句来检索光标指向的下一行，并将光标移动到結果集中的下一行。

```
FETCH cursor_name INTO variables list;
```

之后，可以检查是否有任何行记录可用，然后再提取它。

最后，调用 `CLOSE` 语句来停用光标并释放与之关联的内存，如下所示：

```
CLOSE cursor_name;
```

当光标不再使用时，应该关闭它。

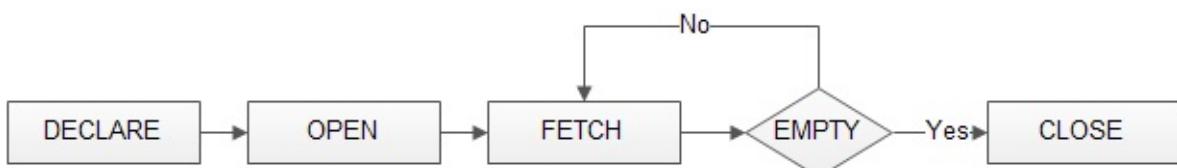
当使用MySQL游标时，还必须声明一个 `NOT FOUND` 处理程序来处理当游标找不到任何行时的情况。因为每次调用 `FETCH` 语句时，游标会尝试读取结果集中的下一行。当光标到达结果集的末尾时，它将无法获得数据，并且会产生一个条件。处理程序用于处理这种情况。

要声明一个 `NOT FOUND` 处理程序，参考以下语法：

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
```

`finished` 是一个变量，指示光标到达结果集的结尾。请注意，处理程序声明必须出现在存储过程中的变量和游标声明之后。

下图说明了MySQL游标如何工作。



MySQL游标示例

为了更好地演示，我们将开发一个存储过程，来获取MySQL示例数据库(`yiibaidb`)中 `employees` 表中所有员工的电子邮件列表。

首先，声明一些变量，一个用于循环员工电子邮件的游标和一个 `NOT FOUND` 处理程序：

```

DECLARE finished INTEGER DEFAULT 0;
DECLARE email varchar(255) DEFAULT "";

-- declare cursor for employee email
DECLARE email_cursor CURSOR FOR
    SELECT email FROM employees;

-- declare NOT FOUND handler
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;

```

接下来，使用 `OPEN` 语句打开 `email_cursor`：

```
OPEN email_cursor;
```

然后，迭代电子邮件列表，并使用分隔符(;)[连接](#)每个电子邮件：

```

get_email: LOOP
    FETCH email_cursor INTO v_email;
    IF v_finished = 1 THEN
        LEAVE get_email;
    END IF;
    -- build email list
    SET email_list = CONCAT(v_email,";",email_list);
END LOOP get_email;

```

之后，在循环中，使用 `v_finished` 变量来检查列表中是否有任何电子邮件来终止循环。

最后，使用 `CLOSE` 语句关闭游标：

```
CLOSE email_cursor;
```

`build_email_list` 存储过程所有代码如下：

```
DELIMITER $$

CREATE PROCEDURE build_email_list (INOUT email_list varchar(4000))
)
BEGIN

DECLARE v_finished INTEGER DEFAULT 0;
    DECLARE v_email varchar(100) DEFAULT "";

-- declare cursor for employee email
DECLARE email_cursor CURSOR FOR
SELECT email FROM employees;

-- declare NOT FOUND handler
DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET v_finished = 1;

OPEN email_cursor;

get_email: LOOP

FETCH email_cursor INTO v_email;

IF v_finished = 1 THEN
    LEAVE get_email;
END IF;

-- build email list
SET email_list = CONCAT(v_email, ";", email_list);

END LOOP get_email;

CLOSE email_cursor;

END$$

DELIMITER ;
```

可以使用以下脚本测试 build_email_list 存储过程：

```
SET @email_list = '';
CALL build_email_list(@email_list);
SELECT @email_list;
```

注：由于内容比较长，这里就不放上输出结果了。

在本教程中，我们向您展示了如何使用MySQL游标来迭代结果集并相应地处理每一行。

在本教程中，我们将向您展示如何列出MySQL数据库中的所有存储过程，并显示存储过程源代码的一些非常有用语句。

MySQL为提供了一些有用的语句，可以更有效地管理存储过程。这些语句包括列出存储过程并显示存储过程的源代码。

显示存储过程字符

要显示存储过程的字符，请使用 `SHOW PROCEDURE STATUS` 语句如下：

```
SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE expr];
```

`SHOW PROCEDURE STATUS` 语句是对SQL标准的MySQL扩展。此语句提供存储过程的字符，包括数据库，存储过程名称，类型，创建者等。

可以使用`LIKE`或`WHERE`子句根据各种标准过滤出存储过程。

要列出您有权访问的数据库的所有存储过程，请使用 `SHOW PROCEDURE STATUS` 语句，如下所示：

```
SHOW PROCEDURE STATUS;
```

如果要在特定数据库中显示存储过程，可以在 `SHOW PROCEDURE STATUS` 语句中使用 `WHERE` 子句：

```
SHOW PROCEDURE STATUS WHERE db = 'yiibaidb';
```

如果要显示具有特定模式的存储过程，例如，名称包含 `product` 字符，则可以使用 `LIKE` 操作符，如以下命令：

```
SHOW PROCEDURE STATUS WHERE name LIKE '%product%'
```

显示存储过程的源代码

要显示特定存储过程的源代码，请使用 `SHOW CREATE PROCEDURE` 语句如下：

```
SHOW CREATE PROCEDURE stored_procedure_name
```

在 `SHOW CREATE PROCEDURE` 关键字之后指定存储过程的名称。例如，要显示 `GetAllProducts` 存储过程的代码，请使用以下语句：

```
SHOW CREATE PROCEDURE GetAllProducts;
```

在本教程中，您已经学习了一些有用的语句，包括 `SHOW PROCEDURE STATUS` 和 `SHOW CREATE PROCEDURE` 语句，用于列出数据库中的存储过程并获取存储过程的源代码。

本教程将向您展示如何使用MySQL处理程序来处理在存储过程中遇到的异常或错误。

当存储过程中发生错误时，重要的是适当处理它，例如：继续或退出当前代码块的执行，并发出有意义的错误消息。

MySQL提供了一种简单的方法来定义处理从一般条件(如警告或异常)到特定条件(例如特定错误代码)的处理程序。

声明处理程序

要声明一个处理程序，您可以使用 `DECLARE HANDLER` 语句如下：

```
DECLARE action HANDLER FOR condition_value statement;
```

如果条件的值与 `condition_value` 匹配，则MySQL将执行 `statement`，并根据该操作继续或退出当前的代码块。

操作(`action`)接受以下值之一：

- `CONTINUE`：继续执行封闭代码块(`BEGIN ... END`)。
- `EXIT`：处理程序声明封闭代码块的执行终止。

`condition_value` 指定一个特定条件或一类激活处理程序的条件。`condition_value` 接受以下值之一：

- 一个MySQL错误代码。
- 标准 `SQLSTATE` 值或者它可以是 `SQLWARNING`，`NOTFOUND` 或 `SQLEXCEPTION` 条件，这是 `SQLSTATE` 值类的简写。`NOTFOUND` 条件用于游标或 `SELECT INTO variable_list` 语句。
- 与MySQL错误代码或 `SQLSTATE` 值相关联的命名条件。

该语句可以是一个简单的语句或由 `BEGIN` 和 `END` 关键字包围的复合语句。

MySQL错误处理示例

我们来看几个声明处理程序的例子。

以下处理程序意味着如果发生错误，则将 `has_error` 变量的值设置为 `1` 并继续执行。

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET has_error = 1;
```

以下是另一个处理程序，如果发生错误，回滚上一个操作，发出错误消息，并退出当前代码块。如果在存储过程的 `BEGIN END` 块中声明它，则会立即终止存储过程。

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
ROLLBACK;
SELECT 'An error has occurred, operation rollbacked and the stored procedure was terminated';
END;
```

以下处理程序如果没有更多的行要提取，在光标或 `SELECT INTO` 语句的情况下，将 `no_row_found` 变量的值设置为 `1` 并继续执行。

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_row_found = 1;
```

以下处理程序如果发生重复的键错误，则会发出 MySQL 错误 `1062`。它发出错误消息并继续执行。

```
DECLARE CONTINUE HANDLER FOR 1062
SELECT 'Error, duplicate key occurred';
```

存储过程中的 MySQL 处理程序示例

首先，为了更好地演示，我们创建一个名为 `article_tags` 的新表：

```
USE testdb;
CREATE TABLE article_tags(
    article_id INT,
    tag_id     INT,
    PRIMARY KEY(article_id, tag_id)
);
```

`article_tags` 表存储文章和标签之间的关系。每篇文章可能有很多标签，反之亦然。为了简单起见，我们不会在 `article_tags` 表中创建文章(`article`)表和标签(`tags`)表以及外键。

接下来，创建一个存储过程，将文章的 `id` 和标签的 `id` 插入到 `article_tags` 表中：

```
USE testdb;
DELIMITER $$

CREATE PROCEDURE insert_article_tags(IN article_id INT, IN tag_id INT)
BEGIN

    DECLARE CONTINUE HANDLER FOR 1062
        SELECT CONCAT('duplicate keys (' ,article_id,' ,',tag_id,) found')
    ) AS msg;

    -- insert a new record into article_tags
    INSERT INTO article_tags(article_id,tag_id)
    VALUES(article_id,tag_id);

    -- return tag count for the article
    SELECT COUNT(*) FROM article_tags;
END$$
DELIMITER ;
```

然后，通过调用 `insert_article_tags` 存储过程，为文章ID为 1 添加标签 ID： 1 , 2 和 3 ，如下所示：

```
CALL insert_article_tags(1,1);
CALL insert_article_tags(1,2);
CALL insert_article_tags(1,3);
```

之后，尝试插入一个重复的键来检查处理程序是否真的被调用。

```
CALL insert_article_tags(1,3);
```

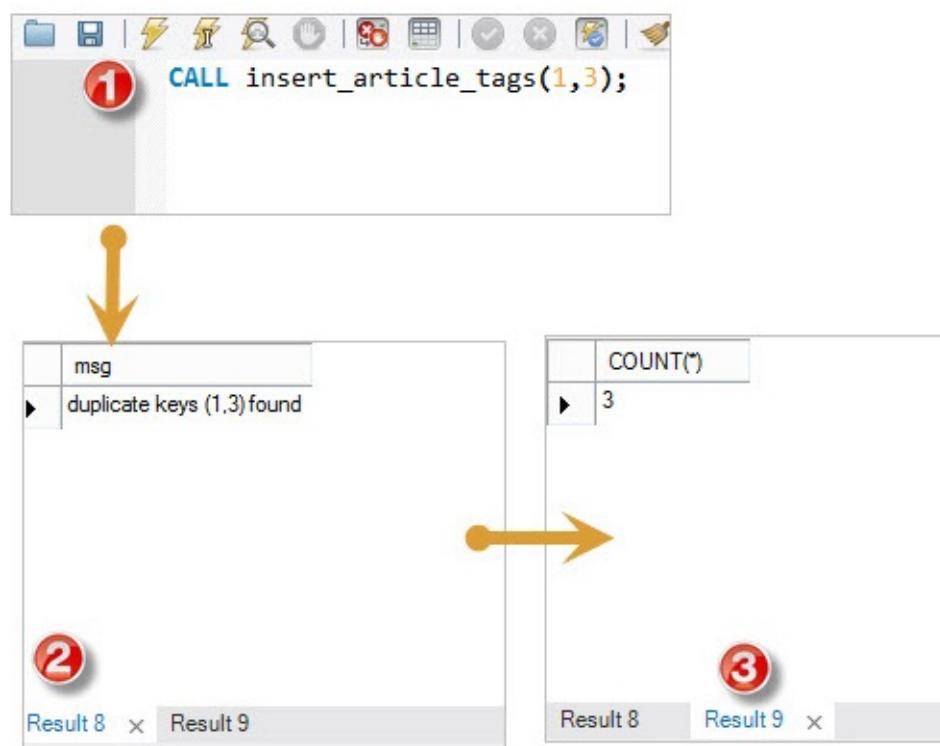
执行上面查询语句，得到以下结果 -

```
mysql> CALL insert_article_tags(1,3);
+-----+
| msg |
+-----+
| duplicate keys (1,3) found |
+-----+
1 row in set

+-----+
| COUNT(*) |
+-----+
|      3   |
+-----+
1 row in set

Query OK, 0 rows affected
```

执行后会收到一条错误消息。但是，由于我们将处理程序声明为 `CONTINUE` 处理程序，所以存储过程继续执行。因此，最后获得了文章的标签计数值为： 3 。



如果将处理程序声明中的 `CONTINUE` 更改为 `EXIT` , 那么将只会收到一条错误消息。如下查询语句 -

```

DELIMITER $$

CREATE PROCEDURE insert_article_tags_exit(IN article_id INT, IN
tag_id INT)
BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION
SELECT 'SQLEXception invoked';

DECLARE EXIT HANDLER FOR 1062
    SELECT 'MySQL error code 1062 invoked';

DECLARE EXIT HANDLER FOR SQLSTATE '23000'
SELECT 'SQLSTATE 23000 invoked';

-- insert a new record into article_tags
INSERT INTO article_tags(article_id,tag_id)
VALUES(article_id,tag_id);

-- return tag count for the article
SELECT COUNT(*) FROM article_tags;
END $$

DELIMITER ;

```

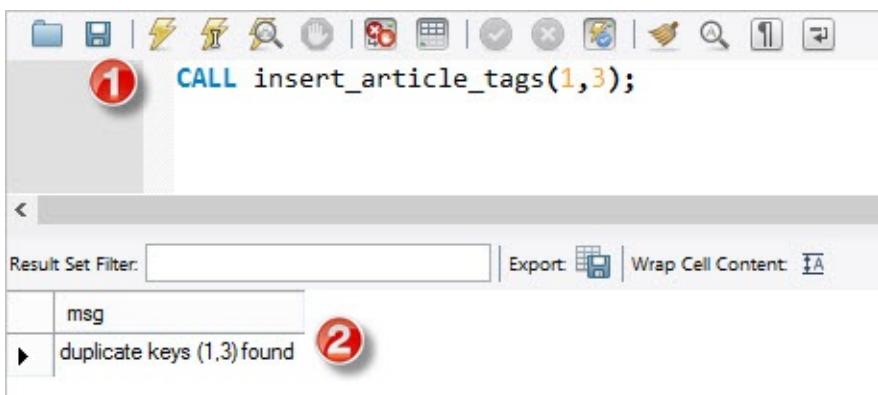
执行上面查询语句，得到以下结果 -

```

mysql> CALL insert_article_tags_exit(1,3);
+-----+
| MySQL error code 1062 invoked |
+-----+
| MySQL error code 1062 invoked |
+-----+
1 row in set

Query OK, 0 rows affected

```



MySQL 处理程序优先级

如果使用多个处理程序来处理错误，MySQL 将调用最特定的处理程序来处理错误。

错误总是映射到一个 MySQL 错误代码，因为在 MySQL 中它是最具体的。

`SQLSTATE` 可以映射到许多 MySQL 错误代码，因此它不太具体。

`SQLEXCEPTION` 或 `SQLWARNING` 是 `SQLSTATES` 类型值的缩写，因此它是最通用的。

假设在 `insert_article_tags_3` 存储过程中声明三个处理程序，如下所示：

```

DELIMITER $$

CREATE PROCEDURE insert_article_tags_3(IN article_id INT, IN tag_id INT)
BEGIN

    DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered';
    DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered';
    DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000';

    -- insert a new record into article_tags
    INSERT INTO article_tags(article_id,tag_id)
    VALUES(article_id,tag_id);

    -- return tag count for the article
    SELECT COUNT(*) FROM article_tags;
END $$

DELIMITER ;

```

我们尝试通过调用存储过程将重复的键插入到 `article_tags` 表中：

```
CALL insert_article_tags_3(1,3);
```

如下，可以看到MySQL错误代码处理程序被调用。

```

mysql> CALL insert_article_tags_3(1,3);
+-----+
| Duplicate keys error encountered |
+-----+
| Duplicate keys error encountered |
+-----+
1 row in set

Query OK, 0 rows affected

```

使用命名错误条件

从错误处理程序声明开始，如下 -

```
DECLARE EXIT HANDLER FOR 1051 SELECT 'Please create table abc first';
SELECT * FROM abc;
```

1051 号是什么意思？想象一下，你有一个大的存储过程代码使用了好多类似这样的数字；这将成为维护代码的噩梦。

幸运的是，MySQL为我们提供了声明一个命名错误条件的 `DECLARE CONDITION` 语句，它与条件相关联。

`DECLARE CONDITION` 语句的语法如下：

```
DECLARE condition_name CONDITION FOR condition_value;
```

`condition_value` 可以是MySQL错误代码，例如： 1015 或 SQLSTATE 值。
`condition_value` 由 `condition_name` 表示。

声明后，可以参考 `condition_name`，而不是参考 `condition_value`。

所以可以重写上面的代码如下：

```
DECLARE table_not_found CONDITION for 1051;
DECLARE EXIT HANDLER FOR table_not_found SELECT 'Please create
table abc first';
SELECT * FROM abc;
```

这段代码比以前的代码显然更可读。

请注意，条件声明必须出现在处理程序或游标声明之前。

在本教程中，您将学习如何使用 `SIGNAL` 和 `RESIGNAL` 语句来引发存储过程中的错误条件。

MySQL SIGNAL 语句

使用 `SIGNAL` 语句在存储的程序(例如存储过程，存储函数，触发器或事件)中向调用者返回错误或警告条件。 `SIGNAL` 语句提供了对返回值(如值和消息 `SQLSTATE`)的信息的控制。

以下说明 `SIGNAL` 语句的语法：

```
SIGNAL SQLSTATE | condition_name;  
SET condition_information_item_name_1 = value_1,  
    condition_information_item_name_1 = value_2, etc;
```

`SIGNAL` 关键字是由 `DECLARE CONDITION` 语句声明的 `SQLSTATE` 值或条件名称。请注意，`SIGNAL` 语句必须始终指定使用 `SQLSTATE` 值定义的 `SQLSTATE` 值或命名条件。

要向调用者提供信息，请使用 `SET` 子句。如果要使用值返回多个条件信息项名称，则需要用逗号分隔每个名称/值对。

`condition_information_item_name` 可以是 `MESSAGE_TEXT`，`MYSQL_ERRORNO`，`CURSOR_NAME` 等。

以下存储过程将订单行项目添加到现有销售订单中。如果订单号码不存在，它会发出错误消息。

```

DELIMITER $$

CREATE PROCEDURE AddOrderItem(in orderNo int,
    in productCode varchar(45),
    in qty int,in price double, in lineNo int )

BEGIN
    DECLARE C INT;

    SELECT COUNT(orderNumber) INTO C
    FROM orders
    WHERE orderNumber = orderNo;

    -- check if orderNumber exists
    IF(C != 1) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Order No not found in orders table';
    END IF;
    -- more code below
    --
END $$
DELIMITER ;

```

首先，它使用传递给存储过程的输入订单号对订单进行计数。第二步，如果订单数不是 1，它会引发 `SQLSTATE 45000` 的错误以及 `orders` 表中不存在订单号的错误消息。

请注意，`45000` 是一个通用 `SQLSTATE` 值，用于说明未处理的用户定义异常。

如果调用存储过程 `AddOrderItem()`，但是传递不存在的订单号，那么将收到一条错误消息。

```
CALL AddOrderItem(10, 'S10_1678', 1, 95.7, 1);
```

执行上面代码，得到以下结果 -

```
mysql> CALL AddOrderItem(10, 'S10_1678', 1, 95.7, 1);
1644 - Order No not found in orders table
```

MySQL RESIGNAL 语句

除了 `SIGNAL` 语句，MySQL 还提供了用于引发警告或错误条件的 `RESIGNAL` 语句。

`RESIGNAL` 语句在功能和语法方面与 `SIGNAL` 语句相似，只是：

- 必须在错误或警告处理程序中使用 `RESIGNAL` 语句，否则您将收到一条错误消息，指出“`RESIGNAL when handler is not active`”。请注意，您可以在存储过程中的任何位置使用 `SIGNAL` 语句。
- 可以省略 `RESIGNAL` 语句的所有属性，甚至可以省略 `SQLSTATE` 值。

如果单独使用 `RESIGNAL` 语句，则所有属性与传递给条件处理程序的属性相同。

以下存储过程在将发送给调用者之前更改错误消息。

```
DELIMITER $$

CREATE PROCEDURE Divide(IN numerator INT, IN denominator INT, OUT result double)
BEGIN
    DECLARE division_by_zero CONDITION FOR SQLSTATE '22012';

    DECLARE CONTINUE HANDLER FOR division_by_zero
        RESIGNAL SET MESSAGE_TEXT = 'Division by zero / Denominator can
not be zero';

    --
    IF denominator = 0 THEN
        SIGNAL division_by_zero;
    ELSE
        SET result := numerator / denominator;
    END IF;
END $$
DELIMITER ;
```

下面我们来调用 Divide() 存储过程。

```
CALL Divide(10, 0, @result);
```

执行上面语句，得到以下结果 -

```
mysql> CALL Divide(10, 0, @result);
1644 - Division by zero / Denominator cannot be zero
```

在本教程中，我们向您展示了如何使用 SIGNAL 和 RESIGNAL 语句引发存储程序中的错误条件。

在本教程中，您将学习如何使用 `CREATE FUNCTION` 语句创建存储的函数。

存储的函数是返回单个值的特殊类型的存储程序。您使用存储的函数来封装在SQL语句或存储的程序中可重用的常用公式或业务规则。

与[存储过程](#)不同，您可以在SQL语句中使用存储的函数，也可以在表达式中使用。这有助于提高程序代码的可读性和可维护性。

MySQL存储函数语法

以下说明了创建新存储函数的最简单语法：

```
CREATE FUNCTION function_name(param1, param2, ...)  
    RETURNS datatype  
    [NOT] DETERMINISTIC  
    statements
```

首先，在`CREATE FUNCTION`子句之后指定存储函数的名称。其次，列出括号内存存储函数的所有[参数](#)。默认情况下，所有参数均为`IN`参数。不能为参数指定`IN`，`OUT`或`INOUT`修饰符。第三，必须在`RETURNS`语句中指定返回值的数据类型。它可以是任何有效的[MySQL数据类型](#)。第四，对于相同的输入参数，如果存储的函数返回相同的结果，这样则被认为是确定性的，否则存储的函数不是确定性的。必须决定一个存储函数是否是确定性的。如果您声明不正确，则存储的函数可能会产生意想不到的结果，或者不使用可用的优化，从而降低性能。第五，将代码写入存储函数的主体中。它可以是单个语句或复合语句。在主体部分中，必须至少指定一个`RETURN`语句。`RETURN`语句用于返回一个值给调用者。每当到达`RETURN`语句时，存储的函数的执行将立即终止。

MySQL存储函数示例

我们来看一下使用存储函数的例子，这里将使用[示例数据库\(yiibaidb\)](#)中的`customers`表进行演示。

以下示例是根据信用额度返回客户级别的功能。我们使用[IF语句](#)来确定信用额度。

```

DELIMITER $$

CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);

    IF p_creditLimit > 50000 THEN
        SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;

    RETURN (lvl);
END $$

DELIMITER ;

```

现在，我们在SELECT语句中调用 CustomerLevel() 存储函数，如下所示：

```

SELECT
    customerName, CustomerLevel(creditLimit)
FROM
    customers
ORDER BY customerName;

```

执行上面查询语句，得到以下结果 -

customerName	CustomerLevel(creditLimit)
Alpha Cognac	PLATINUM

American Souvenirs Inc	SILVER
Amica Models & Co.	PLATINUM
ANG Resellers	SILVER
Anna's Decorations, Ltd	PLATINUM
Anton Designs, Ltd.	SILVER
Asian Shopping Network, Co	SILVER
Asian Treasures, Inc.	SILVER
Atelier graphique	GOLD
Australian Collectables, Ltd	PLATINUM
Australian Collectors, Co.	PLATINUM
***** 此处省略了一大波数据 *****	
Vitachrome Inc.	PLATINUM
Volvo Model Replicas, Co	PLATINUM
Warburg Exchange	SILVER
West Coast Collectables Co.	PLATINUM

-----+-----
--+
122 rows in set

下面，来重写在[MySQL IF语句教程](#)中开发的 `GetCustomerLevel()` 存储过程，如下所示：

```
DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
    IN p_customerNumber INT(11),
    OUT p_customerLevel varchar(10)
)
BEGIN
    DECLARE creditlim DOUBLE;

    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber = p_customerNumber;

    SELECT CUSTOMERLEVEL(creditlim)
    INTO p_customerLevel;
END $$

DELIMITER ;
```

如您所见，`GetCustomerLevel()` 存储过程在使用 `CUSTOMERLEVEL()` 存储函数时可读性更高。

请注意，存储函数仅返回单个值。如果没有包含 `INTO` 子句的 `SELECT` 语句，则将会收到错误。

另外，如果存储的函数包含SQL语句，则不应在其他SQL语句中使用它；否则，存储的函数将减慢查询的速度。

第四章 触发器

在本节中，您将学习如何使用MySQL触发器。根据定义，触发器或数据库触发器是自动执行以响应特定事件的存储程序，例如在表中发生的插入，更新或删除。

数据库触发器是保护MySQL数据库中数据完整性的强大工具。此外，自动执行某些数据库操作(如日志记录，审核等)也很有用。

1. SQL触发器简介

- SQL触发器是存储在数据库目录中的一组SQL语句。每当与表相关联的事件发生时，就会执行或触发SQL触发器，例如插入，更新或删除。参考阅读：<http://www.yiibai.com/mysql/sql-triggers.html>

2. MySQL触发器实现

- 本教程将向您介绍MySQL触发器的实现。此外，还将向您展示MySQL如何在MySQL中存储触发器定义和触发器的限制。参考阅读：<http://www.yiibai.com/mysql/trigger-implementation.html>

3. 在MySQL中创建触发器

- 本教程将向您展示如何在MySQL中创建一个简单的触发器来审核表的更改。我们将详细说明 CREATE TRIGGER 语句的语法。参考阅读：<http://www.yiibai.com/mysql/create-the-first-trigger-in-mysql.html>

4. 为相同的触发事件和动作时间创建多个触发器

- 本教程将向您展示如何为MySQL中的相同触发事件和动作时间创建多个触发器。参考阅读：<http://www.yiibai.com/mysql/create-multiple-triggers-for-the-same-trigger-event-and-action-time.html>

5. 管理MySQL中的触发器

- 您将学习如何管理触发器，包括在MySQL数据库中显示，修改和删除触发器。参考阅读：<http://www.yiibai.com/mysql/managing-trigger-in-mysql.html>

6. 使用MySQL计划事件

- MySQL事件是基于预定义的时间表运行的任务，因此有时它被称为预定事件。

MySQL事件也被称为“时间触发”，因为它是由时间触发的，而不是像触发器这样的表更新作为触发条件。参考阅读：<http://www.yiibai.com/mysql/working-mysql-scheduled-event.html>

7.修改MySQL事件

- 本教程将介绍如何使用 `ALTER EVENT` 语句修改现有的MySQL事件。在本教程之后，您将了解如何修改事件计划，如何启用或禁用事件以及如何重命名事件。参考阅读：<http://www.yiibai.com/mysql/modifying-mysql-events.html>

在本教程中，您将了解MySQL触发器实现。另外，我们将向展示MySQL如何将触发器和触发器的限制存储在MySQL中。

MySQL触发器简介

在MySQL中，触发器是一组SQL语句，当对相关联的表上的数据进行更改时，会自动调用该语句。触发器可以被定义为在INSERT，UPDATE或DELETE语句更改数据之前或之后调用。在MySQL 5.7.2版本之前，每个表最多可以定义六个触发器。

- BEFORE INSERT - 在数据插入表之前被激活触发器。
- AFTER INSERT - 在将数据插入表之后激活触发器。
- BEFORE UPDATE - 在表中的数据更新之前激活触发器。
- AFTER UPDATE - 在表中的数据更新之后激活触发器。
- BEFORE DELETE - 在从表中删除数据之前激活触发器。
- AFTER DELETE - 从表中删除数据之后激活触发器。

但是，从MySQL 5.7.2+版本开始，可以为相同的触发事件和动作时间定义多个触发器。

当使用不使用 INSERT，DELETE 或 UPDATE 语句更改表中数据的语句时，不会调用与表关联的触发器。例如，TRUNCATE语句删除表的所有数据，但不调用与该表相关联的触发器。

有些语句使用了后台的 INSERT 语句，如REPLACE语句或LOAD DATA语句。如果使用这些语句，则调用与表关联的相应触发器。

必须要为与表相关联的每个触发器使用唯一的名称。可以为不同的表定义相同的触发器名称，这是一个很好的做法。

应该使用以下命名约定命名触发器：

(BEFORE | AFTER)_**tableName**_(**INSERT** | **UPDATE** | **DELETE**)

例如，before_order_update 是更新 orders 表中的行数据之前调用的触发器。

以下命名约定与上述一样。

```
tablename_(BEFORE | AFTER)_(INSERT | UPDATE | DELETE)
```

例如，`order_before_update` 与上述 `before_order_update` 触发器相同。

MySQL触发存储

MySQL在数据目录中存储触发器，例如：`/data/yiibaidb/`，并使用名为 `tablename.TRG` 和 `triggername.TRN` 的文件：

- `tablename.TRG` 文件将触发器映射到相应的表。
- `triggername.TRN` 文件包含触发器定义。

可以通过将触发器文件复制到备份文件夹来备份MySQL触发器。也可以使用 [mysqldump工具备份触发器](#)。

MySQL触发限制

MySQL触发器覆盖标准SQL中定义的所有功能。但是，在应用程序中使用它们之前，您应该知道一些限制。

MySQL触发器不能：

- 使用在 `SHOW`，`LOAD DATA`，`LOAD TABLE`，[BACKUP DATABASE](#)，`RESTORE`，`FLUSH` 和 `RETURN` 语句之上。
- 使用隐式或明确提交或回滚的语句，如 `COMMIT`，`ROLLBACK`，`START TRANSACTION`，[LOCK/UNLOCK TABLES](#)，`ALTER`，`CREATE`，`DROP`，[RENAME](#) 等。
- 使用[准备语句](#)，如 `PREPARE`，`EXECUTE` 等
- 使用动态SQL语句。

从MySQL 5.1.4版本开始，触发器可以调用[存储过程](#)或存储函数，在这之前的版本是有所限制的。

在本教程中，我们向您展示了如何在MySQL中实现触发器。我们还讨论了触发器的存储以及触发器在MySQL中的局限性。

在本教程中，您将学习如何使用 `CREATE TRIGGER` 语句在MySQL中创建触发器。

应该先阅读[SQL触发器的简介](#)，并在开始本教程之前先来了解触发MySQL的实现。

MySQL触发语法

为了创建一个新的触发器，可以使用 `CREATE TRIGGER` 语句。下面说明了 `CREATE TRIGGER` 语句的语法：

```
CREATE TRIGGER trigger_name trigger_time trigger_event
  ON table_name
  FOR EACH ROW
  BEGIN
    ...
  END;
```

下面，我们来更详细的检查上面的语法。

- 将触发器名称放在 `CREATE TRIGGER` 语句之后。触发器名称应遵循命名约定 `[trigger time]_[table name]_[trigger event]`，例如 `before_employees_update`。
- 触发激活时间可以在之前或之后。必须指定定义触发器的激活时间。如果要在更改之前处理操作，则使用 `BEFORE` 关键字，如果在更改后需要处理操作，则使用 `AFTER` 关键字。
- 触发事件可以是 `INSERT`，`UPDATE` 或 `DELETE`。此事件导致触发器被调用。触发器只能由一个事件调用。要定义由多个事件调用的触发器，必须定义多个触发器，每个事件一个触发器。
- 触发器必须与特定表关联。没有表触发器将不存在，所以必须在 `ON` 关键字之后指定表名。
- 将SQL语句放在 `BEGIN` 和 `END` 块之间。这是定义触发器逻辑的位置。

MySQL触发器示例

下面我们将在MySQL中创建触发器来记录 `employees` 表中行数据的更改情况。

```
mysql> DESC employees;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| employeeNumber | int(11) | NO | PRI | NULL |
| lastName | varchar(50) | NO | | NULL |
| firstName | varchar(50) | NO | | NULL |
| extension | varchar(10) | NO | | NULL |
| email | varchar(100) | NO | | NULL |
| officeCode | varchar(10) | NO | MUL | NULL |
| reportsTo | int(11) | YES | MUL | NULL |
| jobTitle | varchar(50) | NO | | NULL |
+-----+-----+-----+-----+-----+
8 rows in set
```

首先，创建一个名为 `employees audit` 的新表，用来保存 `employees` 表中数据的更改。以下语句创建 `employee_audit` 表。

```
USE yiibaidb;
CREATE TABLE employees_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employeeNumber INT NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

接下来，创建一个 `BEFORE UPDATE` 触发器，该触发器在对 `employees` 表中的行记录更改之前被调用。

```
DELIMITER $$  
CREATE TRIGGER before_employee_update  
    BEFORE UPDATE ON employees  
    FOR EACH ROW  
BEGIN  
    INSERT INTO employees_audit  
    SET action = 'update',  
        employeeNumber = OLD.employeeNumber,  
        lastname = OLD.lastname,  
        changedat = NOW();  
END$$  
DELIMITER ;
```

在触发器的主体中，使用 `OLD` 关键字来访问受触发器影响的行的 `employeeNumber` 和 `lastname` 列。

请注意，在为 `INSERT` 定义的触发器中，可以仅使用 `NEW` 关键字。不能使用 `OLD` 关键字。但是，在为 `DELETE` 定义的触发器中，没有新行，因此您只能使用 `OLD` 关键字。在 `UPDATE` 触发器中，`OLD` 是指更新前的行，而 `NEW` 是更新后的行。

然后，要查看当前数据库中的所有触发器，请使用 `SHOW TRIGGERS` 语句，如下所示：

```
SHOW TRIGGERS;
```

执行上面查询语句，得到以下结果 -

之后，更新 `employees` 表以检查触发器是否被调用。

```
UPDATE employees
SET
    lastName = 'Maxsu'
WHERE
    employeeNumber = 1056;
```

最后，要检查触发器是否被 `UPDATE` 语句调用，可以使用以下查询来查询 `employees_audit` 表：

```
SELECT * FROM employees_audit;
```

以下是查询的输出：

```
mysql> SELECT * FROM employees_audit;
+-----+-----+-----+-----+-----+
| id | employeeNumber | lastname | changedat | action |
+-----+-----+-----+-----+-----+
| 1 | 1056 | Hill | 2017-08-02 22:15:51 | update |
+-----+-----+-----+-----+-----+
1 row in set
```

如上面输出结果所示，触发器被真正调用，并在 `employees_audit` 表中插入一个新行。

在本教程中，您已经学会了如何在MySQL中创建一个触发器。我们还向您展示了如何开发触发器来审计员工(`employees`)表的更改。

在本教程中，您将学习如何为MySQL中相同的事件和动作时间创建多个触发器。

本教程与MySQL 5.7.2+ 版本相关。如果您有一个较旧版本的MySQL，本教程中的语句将无法正常工作。

在MySQL 5.7.2+ 版本之前，您只能为表中的事件创建一个触发器，例如，只能为 BEFORE UPDATE 或 AFTER UPDATE 事件创建一个触发器。MySQL 5.7.2+ 版本解决了这样限制，并允许您为表中的相同事件和动作时间创建多个触发器。当事件发生时，触发器将依次激活。

参考[创建第一个触发器](#)中的语法。如果表中有相同事件有多个触发器，MySQL 将按照创建的顺序调用触发器。要更改触发器的顺序，需要在 FOR EACH ROW 子句之后指定 FOLLOWES 或 PRECEDES。如下说明 -

- FOLLOWES 选项允许新触发器在现有触发器之后激活。
- PRECEDES 选项允许新触发器在现有触发器之前激活。

以下是使用显式顺序创建新的附加触发器的语法：

```
DELIMITER $$  
CREATE TRIGGER trigger_name  
[BEFORE|AFTER] [INSERT|UPDATE|DELETE] ON table_name  
FOR EACH ROW [FOLLOWES|PRECEDES] existing_trigger_name  
BEGIN  
...  
END$$  
DELIMITER ;
```

MySQL 多重触发器示例

我们来看如何一个在表中的同一个事件和动作上，创建多个触发器的例子。

下面将使用[示例数据库\(yiibaidb\)](#)中的 products 表进行演示。假设，每当更改产品的价格(MSRP 列)时，要将旧的价格记录在一个名为 price_logs 的表中。

首先，使用CREATE TABLE语句创建一个新的 price_logs 表，如下所示：

```

USE yiibaidb;
CREATE TABLE price_logs (
    id INT(11) NOT NULL AUTO_INCREMENT,
    product_code VARCHAR(15) NOT NULL,
    price DOUBLE NOT NULL,
    updated_at TIMESTAMP NOT NULL DEFAULT
        CURRENT_TIMESTAMP
        ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    KEY product_code (product_code),
    CONSTRAINT price_logs_ibfk_1 FOREIGN KEY (product_code)
    REFERENCES products (productCode)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

```

其次，当表的 `BEFORE UPDATE` 事件发生时，创建一个新的触发器。触发器名称为 `before_products_update`，具体实现如下所示：

```

DELIMITER $$

CREATE TRIGGER before_products_update
    BEFORE UPDATE ON products
    FOR EACH ROW
BEGIN
    INSERT INTO price_logs(product_code,price)
    VALUES(old.productCode,old.msrp);
END$$

DELIMITER ;

```

第三，我们更改产品的价格，并使用以下`UPDATE`语句，最后查询 `price_logs` 表：

```
UPDATE products
SET msrp = 95.1
WHERE productCode = 'S10_1678';
-- 查询结果价格记录
SELECT * FROM price_logs;
```

上面查询语句执行后，得到以下结果 -

id	product_code	price	updated_at
1	S10_1678	95.7	2017-08-03 02:46:42

1 row in set

可以看到结果中，它按我们预期那样工作了。

假设不仅要看到旧的价格，改变的时候，还要记录是谁修改了它。我们可以向 `price_logs` 表添加其他列。但是，为了实现多个触发器的演示，我们将创建一个新表来存储进行更改的用户的 data。这个新表的名称为 `user_change_logs`，结构如下：

```

USE yiibaidb;
CREATE TABLE user_change_logs (
    id int(11) NOT NULL AUTO_INCREMENT,
    product_code varchar(15) DEFAULT NULL,
    updated_at timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
    ON UPDATE CURRENT_TIMESTAMP,
    updated_by varchar(30) NOT NULL,
    PRIMARY KEY (id),
    KEY product_code (product_code),
    CONSTRAINT user_change_logs_ibfk_1 FOREIGN KEY (product_code)
    REFERENCES products (productCode)
    ON DELETE CASCADE ON UPDATE CASCADE
);

```

现在，我们创建一个在 `products` 表上的 `BEFORE UPDATE` 事件上激活的第二个触发器。此触发器将更改的用户信息更新到 `user_change_logs` 表。它在 `before_products_update` 触发后被激活。

```

DELIMITER $$

CREATE TRIGGER before_products_update_2
    BEFORE UPDATE ON products
    FOR EACH ROW
        CALL before_products_update();

BEGIN
    INSERT INTO user_change_logs(product_code,updated_by)
    VALUES(old.productCode,user());
END$$

DELIMITER ;

```

下面我们来做一个快速测试。

首先，使用 `UPDATE` 语句更新指定产品的价格，如下：

```
UPDATE products
SET msrp = 95.3
WHERE productCode = 'S10_1678';
```

其次，分别从 `price_logs` 和 `user_change_logs` 表查询数据：

```
SELECT * FROM price_logs;
```

上面查询语句执行后，得到以下结果 -

```
mysql> SELECT * FROM price_logs;
+----+-----+-----+-----+
| id | product_code | price | updated_at |
+----+-----+-----+-----+
| 1 | S10_1678 | 95.7 | 2017-08-03 02:46:42 |
| 2 | S10_1678 | 95.1 | 2017-08-03 02:47:21 |
+----+-----+-----+-----+
2 rows in set
```

```
SELECT * FROM user_change_logs;
```

上面查询语句执行后，得到以下结果 -

```
mysql> SELECT * FROM user_change_logs;
+----+-----+-----+-----+
| id | product_code | updated_at | updated_by |
+----+-----+-----+-----+
| 1 | S10_1678 | 2017-08-03 02:47:21 | root@localhost |
+----+-----+-----+-----+
1 row in set
```

如上所见，两个触发器按照预期的顺序激活执行相关操作了。

触发器顺序

如果使用 `SHOW TRIGGERS` 语句，则不会在表中看到触发激活同一事件和操作的顺序。

```
SHOW TRIGGERS FROM yiibaidb;
```

要查找此信息，需要如下查询 `information_schema` 数据库的 `triggers` 表中的 `action_order` 列，如下查询语句 -

```
SELECT
    trigger_name, action_order
FROM
    information_schema.triggers
WHERE
    trigger_schema = 'yiibaidb'
ORDER BY event_object_table ,
    action_timing ,
    event_manipulation;
```

上面查询语句执行后，得到以下结果 -

```
mysql> SELECT
    trigger_name, action_order
FROM
    information_schema.triggers
WHERE
    trigger_schema = 'yiibaidb'
ORDER BY event_object_table ,
    action_timing ,
    event_manipulation;
+-----+-----+
| trigger_name          | action_order |
+-----+-----+
| before_employee_update |      1       |
| before_products_update |      1       |
| before_products_update_2 |      2       |
+-----+-----+
3 rows in set
```

在本教程中，我们向您展示了如何在MySQL的表中为同一事件创建多个触发器。

在本教程中，您将学习如何管理触发器，包括在MySQL数据库中显示，修改和删除触发器。

创建触发器后，可以在包含触发器定义文件的数据文件夹中显示其定义。触发器作为纯文本文件存储在以下数据库文件夹中：

```
/data_folder/database_name/table_name.trg
```

也可通过查询 `information_schema` 数据库中的 `triggers` 表来显示触发器，如下所示：

```
SELECT
  *
FROM
  information_schema.triggers
WHERE
  trigger_schema = 'database_name'
  AND trigger_name = 'trigger_name';
```

该语句允许您查看触发器的内容及其元数据，例如：关联表名和定义器，这是创建触发器的MySQL用户 的名称。

如果要检索指定数据库中的所有触发器，则需要使用以下SELECT语句从 `information_schema` 数据库中的 `triggers` 表查询数据：

```
SELECT
  *
FROM
  information_schema.triggers
WHERE
  trigger_schema = 'database_name';
```

要查找与特定表相关联的所有触发器，请使用以下查询：

```
SELECT
    *
FROM
    information_schema.triggers
WHERE
    trigger_schema = 'database_name'
        AND event_object_table = 'table_name' ;
```

例如，以下查询语句与 `yiibaidb` 数据库中的 `employees` 表相关联的所有触发器。

```
SELECT * FROM information_schema.triggers  
WHERE trigger_schema = 'yiibaidb'  
      AND event_object_table = 'employees' /
```

执行上面查询，得到以下结果 -

```
mysql> SELECT * FROM information_schema.triggers
WHERE trigger_schema = 'yiibaidb'
AND event_object_table = 'employees';
+-----+-----+-----+
| TRIGGER_CATALOG | TRIGGER_SCHEMA | TRIGGER_NAME | EV
ENT_MANIPULATION | EVENT_OBJECT_CATALOG | EVENT_OBJECT_SCHEMA |
EVENT_OBJECT_TABLE | ACTION_ORDER | ACTION_CONDITION | ACTION_ST
ATEMENT
+-----+-----+-----+
| ACTION_ORIENTATION | ACTION_TIMIN
G | ACTION_REFERENCE_OLD_TABLE | ACTION_REFERENCE_NEW_TABLE | AC
TION_REFERENCE_OLD_ROW | ACTION_REFERENCE_NEW_ROW | CREATED

```

SQL_MODE	DEFINER	CHARACTER_SET_NAME
CLIENT	COLLATION_CONNECTION	DATABASE_COLLATION
def	yiibaidb	before_employee_update
DATE	def	yiibaidb
employees	1	NULL
		BEGIN
	INSERT INTO employees_audit	
	SET action = 'update',	
	employeeNumber = OLD.employeeNumber,	
	lastname = OLD.lastname,	
	changedat = NOW();	
END	ROW	BEFORE
	NULL	OLD
NEW		2017-08-02 22:06:36.40
OUP_BY, STRICT_TRANS_TABLES, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	root@localhost	utf8
		utf8_general_ci
	utf8_general_ci	
1 row in set		

MySQL SHOW TRIGGERS语句

在特定数据库中显示触发器的另一种方法是使用 `SHOW TRIGGERS` 语句，如下所示：

```
SHOW TRIGGERS [FROM|IN] database_name  
[LIKE expr | WHERE expr];
```

例如，如果要查看当前数据库中的所有触发器，可以使用 `SHOW TRIGGERS` 语句，如下所示：

```
SHOW TRIGGERS;
```

要获取特定数据库中的所有触发器，请在 `SHOW TRIGGERS` 语句中指定数据库名称，比如要查询数据库：`yiibaidb` 下的所有触发器，如下所示：

```
SHOW TRIGGERS FROM yiibaidb;
```

上面语句返回 `yiibaidb` 数据库中的所有触发器。

要获取与特定表相关联的所有触发器，可以使用 `SHOW TRIGGERS` 语句中的 `WHERE` 子句。以下语句返回与 `employees` 表相关联的所有触发器：

```
SHOW TRIGGERS FROM yiibaidb  
WHERE `table` = 'employees';
```

请注意，我们使用反引号包装 `table` 列，因为 `table` 是MySQL中的保留关键字。

当执行 `SHOW TRIGGERS` 语句时，MySQL返回以下列 -

- `Trigger` : 存储触发器的名称，例如 `before_employee_update` 触发器。
- `Event` : 指定事件，例如，调用触发器的 `INSERT`，`UPDATE` 或 `DELETE`。
- `Table` : 指定触发器与例如相关联的表,如 `employees` 表。
- `Statement` : 存储调用触发器时要执行的语句或复合语句。

- `Timing` : 接受两个值：`BEFORE` 和 `AFTER`，它指定触发器的激活时间。
- `Created` : 在创建触发器时记录创建的时间。
- `sql_mode` : 指定触发器执行时的SQL模式。
- `Definer` : 记录创建触发器的帐户。

请注意，要执行 `SHOW TRIGGERS` 语句，您必须具有 `SUPER` 权限。

删除触发器

要删除现有的触发器，请使用 `DROP TRIGGER` 语句，如下所示：

```
DROP TRIGGER table_name.trigger_name;
```

例如，如果要删除与 `employees` 表相关联的 `before_employees_update` 触发器，则可以执行以下语句：

```
DROP TRIGGER employees.before_employees_update;
```

要修改触发器，必须首先删除它并使用新的代码重新创建。在MySQL中没有类似: `ALTER TRIGGER` 语句，因此，您不能像修改其他数据库对象，如表，视图和存储过程那样修改触发器。

在本教程中，您已经学会了如何管理触发器，如在MySQL中显示，删除和修改触发器。

在本教程中，您将了解MySQL事件调度程序以及如何创建MySQL事件以自动执行数据库任务。

MySQL事件是基于预定义的时间表运行的任务，因此有时它被称为预定事件。

MySQL事件也被称为“时间触发”，因为它是由时间触发的，而不是像[触发器](#)一样更新表来触发的。MySQL事件类似于UNIX中的cron作业或Windows中的任务调度程序。

您可以在许多情况下使用MySQL事件，例如优化数据库表，清理日志，归档数据或在非高峰时间生成复杂的报告。

MySQL事件调度器配置

MySQL使用一个名为事件调度线程的特殊线程来执行所有调度的事件。可以通过执行以下命令来查看事件调度程序线程的状态：

```
SHOW PROCESSLIST;
```

执行上面查询语句，得到以下结果 -

```
mysql> SHOW PROCESSLIST;
+-----+-----+-----+-----+-----+-----+
| Id | User | Host           | db   | Command | Time | Stat
e   | Info |
+-----+-----+-----+-----+-----+-----+
| 2 | root | localhost:50405 | NULL | Sleep   | 1966 | 
| NULL |
| 3 | root | localhost:50406 | yiibaidb | Sleep  | 1964 | 
| NULL |
| 4 | root | localhost:50407 | yiibaidb | Query  | 0     | star
ting | SHOW PROCESSLIST |
+-----+-----+-----+-----+-----+-----+
3 rows in set
```

默认情况下，事件调度程序线程未启用。要启用和启动事件调度程序线程，需要执行以下命令：

```
SET GLOBAL event_scheduler = ON;
```

现在看到事件调度器线程的状态，再次执行 `SHOW PROCESSLIST` 命令，结果如下所示 -

```
mysql> SHOW PROCESSLIST;
+-----+-----+-----+-----+-----+
| Id   | User      | Host     | db    | Command |
| Time | State     |          | Info  |          |
+-----+-----+-----+-----+-----+
| 2   | root      | localhost:50405 | NULL | Sleep   |
| 1986 |           |                 | NULL  |          |
| 3   | root      | localhost:50406 | yiibaidb | Sleep   |
| 1984 |           |                 | NULL  |          |
| 4   | root      | localhost:50407 | yiibaidb | Query   |
| 0   | starting  |                 | SHOW PROCESSLIST |          |
| 5   | event_scheduler | localhost | NULL | Daemon |
| 6   | Waiting on empty queue | NULL |          |
+-----+-----+-----+-----+-----+
4 rows in set
```

要禁用并停止事件调度程序线程，可通过执行 `SET GLOBAL` 命令将 `event_scheduler` 其值设置为 `OFF`：

```
SET GLOBAL event_scheduler = OFF;
```

创建新的MySQL事件

创建事件与创建其他数据库对象(如存储过程或触发器)类似。事件是一个包含SQL语句的命名对象。

存储过程仅在直接调用时执行；触发器则与一个表相关联的事件（例如插入，更新或删除）事件发生时，可以在一次或更多的规则间隔执行事件时执行触发。

要创建和计划新事件，请使用 `CREATE EVENT` 语句，如下所示：

```
CREATE EVENT [IF NOT EXIST] event_name  
ON SCHEDULE schedule  
DO  
event_body
```

下面让我们更详细地解释语法中的一些参数 -

- 首先，在 `CREATE EVENT` 子句之后指定事件名称。事件名称在数据库模式中必须是唯一的。
- 其次，在 `ON SCHEDULE` 子句后面加上一个表。如果事件是一次性事件，则使用语法：`AT timestamp [+ INTERVAL]`，如果事件是循环事件，则使用 `EVERY` 子句：`EVERY interval STARTS timestamp [+INTERVAL] ENDS timestamp [+INTERVAL]`
- 第三，将 `DO` 语句放在 `DO` 关键字之后。请注意，可以在事件主体内调用存储过程。如果您有复合SQL语句，可以将它们放在 `BEGIN END` 块中。

我们来看几个创建事件的例子来了解上面的语法。

首先，创建并计划将一个消息插入到 `messages` 表中的一次性事件，请执行以下步骤：

```
USE testdb;  
CREATE TABLE IF NOT EXISTS messages (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    message VARCHAR(255) NOT NULL,  
    created_at DATETIME NOT NULL  
);
```

其次，使用 `CREATE EVENT` 语句创建一个事件：

```
CREATE EVENT IF NOT EXISTS test_event_01
ON SCHEDULE AT CURRENT_TIMESTAMP
DO
    INSERT INTO messages(message,created_at)
    VALUES('Test MySQL Event 1',NOW());
```

第三，检查 `messages` 表；会看到有 1 条记录。这意味着事件在创建时被执行。

```
SELECT * FROM messages;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT * FROM messages;
+----+-----+-----+
| id | message          | created_at        |
+----+-----+-----+
| 1  | Test MySQL Event 1 | 2017-08-03 04:23:11 |
+----+-----+-----+
1 row in set
```

要显示数据库(`testdb`)的所有事件，请使用以下语句：

```
SHOW EVENTS FROM testdb;
```

执行上面查询看不到任何行返回，因为事件在到期时自动删除。在我们的示例中，它是一次性的事件，在执行完成时就过期了。

要更改此行为，可以使用 `ON COMPLETION PRESERVE` 子句。以下语句创建另一个一次性事件，在其创建时间 1 分钟后执行，执行后不会被删除。

```
CREATE EVENT test_event_02
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 MINUTE
ON COMPLETION PRESERVE
DO
    INSERT INTO messages(message,created_at)
    VALUES('Test MySQL Event 2',NOW());
```

计划事件

等待 1 分钟后，查看 `messages` 表，添加了另一条记录：

```
SELECT * FROM messages;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT * FROM messages;
+----+-----+-----+
| id | message          | created_at        |
+----+-----+-----+
| 1  | Test MySQL Event 1 | 2017-08-03 04:23:11 |
| 2  | Test MySQL Event 2 | 2017-08-03 04:24:48 |
+----+-----+-----+
2 rows in set
```

如果再次执行 `SHOW EVENTS` 语句，看到事件是由于 `ON COMPLETION PRESERVE` 子句的影响：

```
SHOW EVENTS FROM testdb;
```

执行上面查询语句，得到以下结果 -

```
mysql> SHOW EVENTS FROM testdb;
+-----+-----+-----+-----+
| Db   | Name      | Definer    | Time zone | Type
| Execute at | Interval value | Interval field | Starts
| Ends | Status     | Originator | character_set_client | collatio
n_connection | Database Collation |
+-----+-----+-----+-----+
| testdb | test_event_02 | root@localhost | SYSTEM | ONE TIME
| 2017-08-03 04:24:48 | NULL | NULL | NULL
| NULL | DISABLED | 0 | utf8
| general_ci | utf8_general_ci |
+-----+-----+-----+-----+
1 row in set
```

以下语句创建一个循环的事件，每分钟执行一次，并在其创建时间的 1 小时内过期：

```
CREATE EVENT test_event_03
ON SCHEDULE EVERY 1 MINUTE
STARTS CURRENT_TIMESTAMP
ENDS CURRENT_TIMESTAMP + INTERVAL 1 HOUR
DO
    INSERT INTO messages(message, created_at)
    VALUES('Test MySQL recurring Event', NOW());
```

请注意，使用 `STARTS` 和 `ENDS` 子句定义事件的有效期。等待个3，5分钟后再查看 `messages` 表数据，以测试验证此循环事件的执行。

```
SELECT * FROM messages;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT * FROM messages;
+----+-----+-----+
| id | message | created_at |
+----+-----+-----+
| 1  | Test MySQL Event 1 | 2017-08-03 04:23:11 |
| 2  | Test MySQL Event 2 | 2017-08-03 04:24:48 |
| 3  | Test MySQL recurring Event | 2017-08-03 04:25:20 |
| 4  | Test MySQL recurring Event | 2017-08-03 04:26:20 |
| 5  | Test MySQL recurring Event | 2017-08-03 04:27:20 |
+----+-----+-----+
5 rows in set
```

删除MySQL事件

要删除现有事件，请使用 `DROP EVENT` 语句，如下所示：

```
DROP EVENT [IF EXISTS] event_name;
```

例如，要删除 `test_event_03` 的事件，请使用以下语句：

```
DROP EVENT IF EXISTS test_event_03;
```

在本教程中，您已经了解了MySQL事件，如何从数据库模式创建和删除事件。在下一个教程中，我们将向您展示如何修改事件。

本教程将向您展示如何使用 `ALTER EVENT` 语句修改现有的MySQL事件。在学习完本教程之后，您将了解如何修改事件和计划，如何启用或禁用事件以及如何重命名事件。

MySQL允许您更改现有事件的各种属性。要更改现有事件，请使用 `ALTER EVENT` 语句，如下所示：

```
ALTER EVENT event_name
  ON SCHEDULE schedule
  ON COMPLETION [NOT] PRESERVE
  RENAME TO new_event_name
  ENABLE | DISABLE
  DO
    event_body
```

请注意，`ALTER EVENT` 语句仅适用于存在的事件。如果您尝试修改不存在的事件，MySQL将会发出一条错误消息，因此在更改事件之前，应先使用 `SHOW EVENTS` 语句检查事件的存在。

```
SHOW EVENTS FROM testdb;
```

执行上面查询，得到以下结果 -

```
mysql> SHOW EVENTS FROM testdb;
+-----+-----+-----+-----+
| Db   | Name      | Definer    | Time zone | Type
| Execute at | Interval value | Interval field | Starts
| Ends | Status     | Originator | character_set_client | collatio
n_connection | Database Collation |
+-----+-----+-----+-----+
| testdb | test_event_02 | root@localhost | SYSTEM | ONE TIME
| 2017-08-03 04:24:48 | NULL | NULL | NULL
| NULL | DISABLED | 0 | utf8 | utf8_general_ci
|           | utf8_general_ci |
+-----+-----+-----+-----+
1 row in set
```

ALTER EVENT示例

我们创建一个示例事件来演示如何使用 ALTER EVENT 语句的各种功能。

以下语句创建一个事件，每分钟将一条新记录插入到 messages 表中。

```
USE testdb;
CREATE EVENT test_event_04
ON SCHEDULE EVERY 1 MINUTE
DO
    INSERT INTO messages(message,created_at)
    VALUES('Test ALTER EVENT statement',NOW());
```

改变调度时间

修改事件

要修改事件为每 2 分钟运行一次，请使用以下语句：

```
ALTER EVENT test_event_04  
ON SCHEDULE EVERY 2 MINUTE;
```

改变事件的主体代码逻辑

您还可以通过指定新的逻辑来更改事件的主体代码，如下所示：

```
ALTER EVENT test_event_04  
DO  
    INSERT INTO messages(message,created_at)  
    VALUES('Message from event',NOW());  
    -- 清空表中的数据  
    truncate messages;
```

上面修改完成后，可以等待 2 分钟，再次查看 messages 表：

```
SELECT * FROM messages;
```

执行上面查询，得到以下结果 -

```
mysql> SELECT * FROM messages;  
+----+-----+-----+  
| id | message          | created_at      |  
+----+-----+-----+  
| 1  | Message from event | 2017-08-03 04:46:47 |  
| 2  | Message from event | 2017-08-03 04:48:47 |  
+----+-----+-----+  
2 rows in set
```

禁用事件

要禁用某个事件，请在 ALTER EVENT 语句之后使用 DISABLE 关键字，请使用以下语句：

```
ALTER EVENT test_event_04  
DISABLE;
```

也可以通过使用 `SHOW EVENTS` 语句来查看事件的状态，如下所示：

```
SHOW EVENTS FROM testdb;
```

执行上面查询，得到以下结果 -

```
mysql> SHOW EVENTS FROM testdb;  
+-----+-----+-----+-----+-----+  
| Db   | Name      | Definer    | Time zone | Type  |  
| Execute at | Interval value | Interval field | Star  
ts       | Ends | Status     | Originator | character_set_c  
lient | collation_connection | Database Collation |  
+-----+-----+-----+-----+-----+  
| testdb | test_event_02 | root@localhost | SYSTEM    | ONE TIME  
| 2017-08-03 04:24:48 | NULL        | NULL      | NULL  
| NULL | DISABLED | 0 | utf8  
| utf8_general_ci | utf8_general_ci |  
| testdb | test_event_04 | root@localhost | SYSTEM    | RECURRIN  
G | NULL       | 2          | MINUTE    | 2017-  
08-03 04:44:47 | NULL | DISABLED | 0 | utf8  
| utf8_general_ci | utf8_general_ci |  
+-----+-----+-----+-----+-----+  
2 rows in set
```

启用事件

要启用已禁用的事件，请在 `ALTER EVENT` 语句之后使用 `ENABLE` 关键字，如下所示：

```
ALTER EVENT test_event_04
ENABLE;
```

查询上面语句执行结果，得到以下结果 -

```
mysql> SHOW EVENTS FROM testdb;
+-----+-----+-----+-----+
| Db   | Name      | Definer    | Time zone | Type
| Execute at | Interval value | Interval field | Star
ts          | Ends | Status     | Originator | character_set_c
lient | collation_connection | Database Collation |
+-----+-----+-----+-----+
| testdb | test_event_02 | root@localhost | SYSTEM    | ONE TIME
| 2017-08-03 04:24:48 | NULL        | NULL       | NULL
|           | NULL        | DISABLED    | 0          | utf8
|           | utf8_general_ci | utf8_general_ci | |
| testdb | test_event_04 | root@localhost | SYSTEM    | RECURRIN
G | NULL        | 2            | MINUTE    | 2017-
08-03 04:44:47 | NULL        | ENABLED    | 0          | utf8
|           | utf8_general_ci | utf8_general_ci | |
+-----+-----+-----+-----+
2 rows in set
```

重命名事件

MySQL不提供类似 `RENAME EVENT` 语句。幸运的是，我们可以使用 `ALTER EVENT` 重命名现有事件，如下所示：

```
ALTER EVENT test_event_04
RENAME TO test_event_05;
```

查询上面语句执行结果，得到以下结果 -

```
mysql> SHOW EVENTS FROM testdb;
+-----+-----+-----+-----+
| Db   | Name      | Definer    | Time zone | Type
| Execute at | Interval value | Interval field | Star
ts       | Ends | Status     | Originator | character_set_c
lient | collation_connection | Database Collation |
+-----+-----+-----+-----+
| testdb | test_event_02 | root@localhost | SYSTEM | ONE TIME
| 2017-08-03 04:24:48 | NULL | NULL | NULL
| NULL | DISABLED | 0 | utf8
| utf8_general_ci | utf8_general_ci |
| testdb | test_event_05 | root@localhost | SYSTEM | RECURRIN
G | NULL | 2 | MINUTE | 2017-
08-03 04:44:47 | NULL | ENABLED | 0 | utf8
| utf8_general_ci | utf8_general_ci |
+-----+-----+-----+-----+
2 rows in set
```

将事件移动到其他数据库

可以通过使用 `RENAME TO` 子句将事件从一个数据库移动到另一个数据库中，如下所示：

```
ALTER EVENT testdb.test_event_05
RENAME TO newdb.test_event_05;
```

查询上面语句执行结果，得到以下结果 -

```
mysql> SHOW EVENTS FROM newdb;
+-----+-----+-----+-----+
| Db   | Name      | Definer    | Time zone | Type
| Execute at | Interval value | Interval field | Starts
| Ends | Status | Originator | character_set_client | collation_connection | Database Collation
+-----+-----+-----+-----+
| newdb | test_event_05 | root@localhost | SYSTEM     | RECURRING
| NULL | 2           | MINUTE      | 2017 08 03 04:44
| :47  | NULL        | ENABLED     | 0          | utf8
| general_ci | utf8_general_ci |           |
+-----+-----+-----+-----+
1 row in set
```

假设 newdb 数据库在MySQL数据库服务器中可用。

在本教程中，我们向您展示了如何使用 ALTER EVENT 语句更改MySQL事件的各种属性。

第五章 视图

数据库视图被称为“虚拟表”，允许您查询其中的数据。了解数据库视图并正确使用它们是非常重要的。在本节中，我们将讨论数据库视图，如何在MySQL中实现它们以及如何更有效地使用它们。

1. 数据库视图简介

在本教程中，您将了解什么是数据库视图及其作用。我们将讨论使用数据库视图的优缺点。请参考阅读：<http://www.yiibai.com/mysql/introduction-sql-views.html>

2. MySQL 中的视图

在本教程中，您将了解MySQL中的视图，并演示如何使用和实现MySQL视图。请参考阅读：<http://www.yiibai.com/mysql/views-in-mysql.html>

3. 在 MySQL 中创建视图

本教程将介绍如何使用 `CREATE VIEW` 语句在MySQL中创建视图。请参考阅读：<http://www.yiibai.com/mysql/create-sql-views-mysql.html>

4. 创建 MySQL 可更新视图

MySQL视图不仅可查询，而且可更新。在本教程中，您将学习如何创建可更新视图，并使用`INSERT`，`UPDATE`，`DELETE`语句来更新数据。请参考阅读：<http://www.yiibai.com/mysql/create-sql-updatable-views.html>

5. 使用检查选项确保视图一致性子句

在本教程中，您将学习如何使用 `WITH CHECK OPTION` 子句确保视图的一致性。请参考阅读：<http://www.yiibai.com/mysql/view-with-check-option.html>

6. 了解 LOCAL 和 CASCDED 在检查选项子句

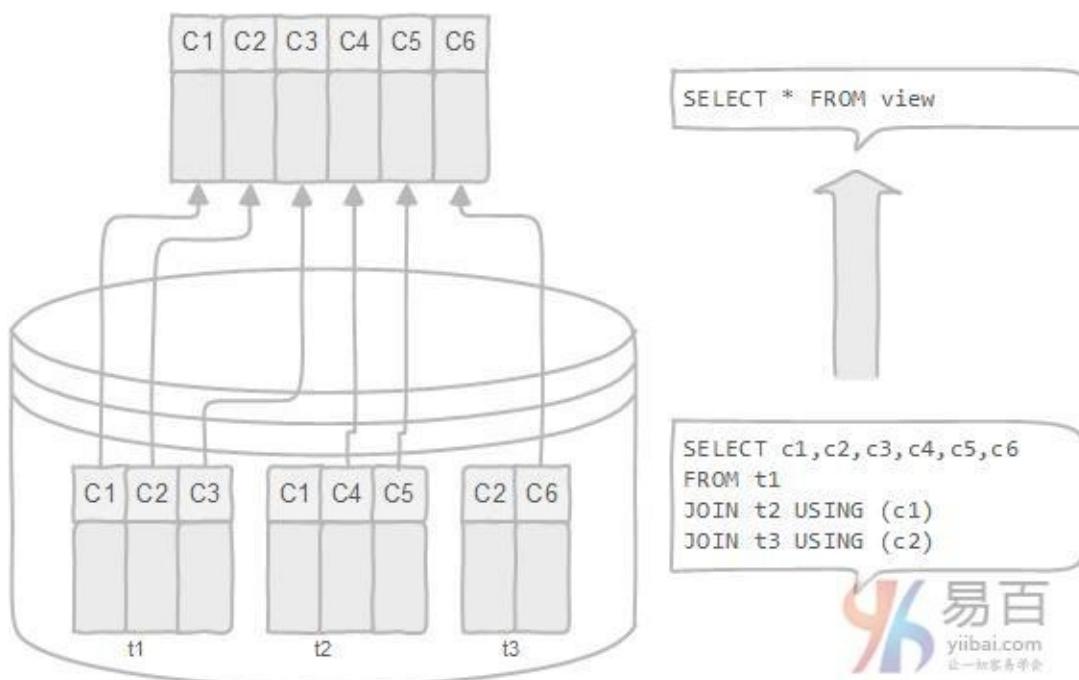
本教程通过示例和明确的说明帮助您了解 `WITH CHECK OPTION` 子句中 `LOCAL` 和 `CASCDED` 之间的差异。请参考阅读：<http://www.yiibai.com/mysql/view-local-cascaded-in-with-check-option.html>

7. 管理 MySQL 中的视图

本教程将向您展示如何管理MySQL中的视图，包括显示，修改和删除视图。请参考阅读：<http://www.yiibai.com/mysql/managing-sql-views.html>

在本教程中，您将了解一个叫作数据库视图的新数据库对象。我们将讨论使用数据库视图的优缺点。

数据库视图是虚拟表或逻辑表，它被定义为具有连接的SQL **SELECT**查询语句。因为数据库视图与数据库表类似，它由行和列组成，因此可以根据数据库表查询数据。大多数数据库管理系统(包括MySQL)允许您通过具有一些先决条件的数据库视图来更新基础表中的数据。



数据库视图是动态的，因为它与物理模式无关。数据库系统将数据库视图存储为具有连接的SQL **SELECT**语句。当表的数据发生变化时，视图也反映了这些数据的变化。

数据库视图的优点

以下是使用数据库视图的优点 -

- 数据库视图允许简化复杂查询：数据库视图由与许多基础表相关联的SQL语句定义。您可以使用数据库视图来隐藏最终用户和外部应用程序的基础表的复杂性。通过数据库视图，您只需使用简单的SQL语句，而不是使用具有多个连接的复杂的SQL语句。
- 数据库视图有助于限制对特定用户的访问。您可能不希望所有用户都可以查询敏感数据的子集。可以使用数据库视图将非敏感数据仅显示给特定用户组。
- 数据库视图提供额外的安全层。安全是任何关系数据库管理系统的重要组成部分。

分。数据库视图为数据库管理系统提供了额外的安全性。数据库视图允许您创建只读视图，以将只读数据公开给特定用户。用户只能以只读视图检索数据，但无法更新。

- 数据库视图启用计算列。数据库表不应该具有计算列，但数据库视图可以这样。假设在 `orderDetails` 表中有 `quantityOrder` (产品的数量) 和 `priceEach` (产品的价格) 列。但是，`orderDetails` 表没有一个列用来存储订单的每个订单项的总销售额。如果有，数据库模式不是一个好的设计。在这种情况下，您可以创建一个名为 `total` 的计算列，该列是 `quantityOrder` 和 `priceEach` 的乘积，以表示计算结果。当您从数据库视图中查询数据时，计算列的数据将随机计算产生。
- 数据库视图实现向后兼容。假设你有一个中央数据库，许多应用程序正在使用它。有一天，您决定重新设计数据库以适应新的业务需求。删除一些表并创建新的表，并且不希望更改影响其他应用程序。在这种情况下，可以创建与将要删除的旧表相同的模式的数据库视图。

数据库视图的缺点

除了上面的优点，使用数据库视图有几个缺点：

- 性能：从数据库视图查询数据可能会很慢，特别是如果视图是基于其他视图创建的。
- 表依赖关系：将根据数据库的基础表创建一个视图。每当更改与其相关联的表的结构时，都必须更改视图。

在本教程中，您已经了解了数据库视图是什么。我们还介绍了使用数据库视图的优点和缺点，以便您可以在数据库设计中有效地应用数据库视图。

在本教程中，您将了解MySQL视图，我们将解释并演示MySQL如何实现视图。

MySQL 5.x 版本之后支持数据库视图。在MySQL中，视图的几乎特征符合SQL：2003标准。MySQL以两种方式处理对视图的查询：

- 第一种方式，MySQL会根据视图定义语句创建一个临时表，并在此临时表上执行传入查询。
- 第二种方式，MySQL将传入查询与查询定义为一个查询并执行组合查询。

MySQL支持版本系统的视图。每次视图被更改或替换时，视图的副本将在驻留在特定数据库文件夹的 arc (archive)文件夹中备份。备份文件的名称为 view_name.frm-00001。如果再次更改视图，MySQL将创建一个名为 view_name.frm-00002 的新备份文件。

MySQL允许基于其他视图创建视图。在视图定义的SELECT语句中，可以引用另一个视图。

MySQL视图的限制

不能在视图上创建索引。当使用合并算法的视图查询数据时，MySQL会使用底层表的索引。对于使用诱惑算法的视图，当您针对视图查询数据时，不会使用索引。

在MySQL 5.7.7之前版本，是不能在 SELECT 语句的 FROM 子句中使用子查询来定义视图的。

如果删除或重命名视图所基于的表，则MySQL不会发出任何错误。但是，MySQL会使视图无效。可以使用 CHECK TABLE 语句来检查视图是否有效。

一个简单的视图可以更新表中数据。基于具有连接，子查询等的复杂SELECT语句创建的视图无法更新。

MySQL不像：Oracle，PostgreSQL等其他数据库系统那样支持物理视图，MySQL是不支持物理视图的。

在本教程中，您将学习如何使用 `CREATE VIEW` 语句在MySQL中创建视图。

CREATE VIEW语句简介

要在MySQL中创建一个新视图，可以使用 `CREATE VIEW` 语句。在MySQL中创建视图的语法如下：

```
CREATE
  [ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]
VIEW [database_name].[view_name]
AS
[SELECT statement]
```

下面我们来详细的查看上面的语法。

查看处理算法

算法属性允许您控制MySQL在创建视图时使用的机制，MySQL提供了三种算法：`MERGE`，`TEMPTABLE` 和 `UNDEFINED`。

- 使用 `MERGE` 算法，MySQL首先将输入查询与定义视图的 `SELECT` 语句组合成单个查询。然后MySQL执行组合查询返回结果集。如果 `SELECT` 语句包含集合函数(如 `MIN`，`MAX`，`SUM`，`COUNT`，`AVG` 等)或 `DISTINCT`，`GROUP BY`，`HAVING`，`LIMIT`，`UNION`，`UNION ALL`，子查询，则不允许使用 `MERGE` 算法。如果 `SELECT` 语句无引用表，则也不允许使用 `MERGE` 算法。如果不允许 `MERGE` 算法，MySQL将算法更改为 `UNDEFINED`。请注意，将视图定义中的输入查询和查询组合成一个查询称为视图分辨率。
- 使用 `TEMPTABLE` 算法，MySQL首先根据定义视图的 `SELECT` 语句创建一个临时表，然后针对该临时表执行输入查询。因为MySQL必须创建临时表来存储结果集并将数据从基表移动到临时表，所以 `TEMPTABLE` 算法的效率比 `MERGE` 算法效率低。另外，使用 `TEMPTABLE` 算法的视图是不可更新的。
- 当您创建视图而不指定显式算法时，`UNDEFINED` 是默认算法。
`UNDEFINED` 算法使MySQL可以选择使用 `MERGE` 或 `TEMPTABLE` 算法。MySQL优先使用 `MERGE` 算法进行 `TEMPTABLE` 算法，因为 `MERGE` 算法效率更高。

查看名称

在数据库中，视图和表共享相同的命名空间，因此视图和表不能具有相同的名称。另外，视图的名称必须遵循表的命名规则。

SELECT语句

在 `SELECT` 语句中，可以从数据库中存在的任何表或视图查询数据。`SELECT` 语句必须遵循以下几个规则：

- `SELECT` 语句可以在 `WHERE子句` 中包含子查询，但 `FROM 子句` 中的不能包含子查询。
- `SELECT` 语句不能引用任何变量，包括局部变量，用户变量和会话变量。
- `SELECT` 语句不能引用准备语句的参数。

请注意，`SELECT` 语句不需要引用任何表。

创建MySQL视图示例

创建简单的视图

我们来看看 `orderDetails` 表。基于 `orderDetails` 表来创建一个表示每个订单的总销售额的视图。

```
CREATE VIEW SalePerOrder AS
  SELECT
    orderNumber, SUM(quantityOrdered * priceEach) total
  FROM
    orderDetails
  GROUP by orderNumber
  ORDER BY total DESC;
```

如果使用 `SHOW TABLES` 命令来查看示例数据库(`yiibaidb`)中的所有表，我们还会看到 `SalesPerOrder` 视图也显示在表的列表中。如下所示 -

```
mysql> SHOW TABLES;
+-----+
| Tables_in_yiibaidb |
+-----+
| article_tags
| contacts
| customers
| departments
| employees
| offices
| offices_bk
| offices_usa
| orderdetails
| orders
| payments
| productlines
| products
| saleperorder
+-----+
14 rows in set
```

这是因为视图和表共享相同的命名空间。要知道哪个对象是视图或表，请使用 `SHOW FULL TABLES` 命令，如下所示：

```
mysql> SHOW FULL TABLES;
+-----+-----+
| Tables_in_yiibaidb | Table_type |
+-----+-----+
| article_tags       | BASE TABLE |
| contacts           | BASE TABLE |
| customers          | BASE TABLE |
| departments         | BASE TABLE |
| employees           | BASE TABLE |
| offices             | BASE TABLE |
| offices_bk          | BASE TABLE |
| offices_usa         | BASE TABLE |
| orderdetails        | BASE TABLE |
| orders              | BASE TABLE |
| payments             | BASE TABLE |
| productlines        | BASE TABLE |
| products             | BASE TABLE |
| saleperorder         | VIEW      |
+-----+-----+
14 rows in set
```

结果集中的 `table_type` 列指定哪个对象是视图，哪个对象是一个表(基表)。如上所示，`saleperorder` 对应 `table_type` 列的值为：`VIEW`。

如果要查询每个销售订单的总销售额，只需要对 `SalePerOrder` 视图执行一个简单的SELECT语句，如下所示：

```
SELECT
  *
FROM
  salePerOrder;
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| orderNumber | total |
+-----+-----+
| 10165 | 67392.85 |
| 10287 | 61402.00 |
| 10310 | 61234.67 |
| 10212 | 59830.55 |
-- 此处省略了一大波数据 --
| 10116 | 1627.56 |
| 10158 | 1491.38 |
| 10144 | 1128.20 |
| 10408 | 615.45 |
+-----+
327 rows in set
```

基于另一个视图创建视图

MySQL允许您基于另一个视图创建一个视图。例如，可以根据 `SalesPerOrder` 视图创建名为大销售订单(`BigSalesOrder`)的视图，以显示总计大于 `60,000` 的每个销售订单，如下所示：

```
CREATE VIEW BigSalesOrder AS
SELECT
    orderNumber, ROUND(total,2) as total
FROM
    saleperorder
WHERE
    total > 60000;
```

现在，我们可以从 `BigSalesOrder` 视图查询数据，如下所示：

```
SELECT
    orderNumber, total
FROM
    BigSalesOrder;
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| orderNumber | total |
+-----+-----+
| 10165 | 67392.85 |
| 10287 | 61402.00 |
| 10310 | 61234.67 |
+-----+
3 rows in set
```

使用连接表创建视图

以下是使用 **INNER JOIN** 创建视图的示例。该视图包含客户编号和客户支付的总金额。

```
CREATE VIEW customerOrders AS
SELECT
    c.customerNumber,
    p.amount
FROM
    customers c
        INNER JOIN
    payments p ON p.customerNumber = c.customerNumber
GROUP BY c.customerNumber
ORDER BY p.amount DESC;
```

要查询 `customerOrders` 视图中的数据，请使用以下查询：

```
SELECT * FROM customerOrders;
```

执行上面查询语句，得到以下结果 -

```
+-----+-----+
| customerNumber | amount |
+-----+-----+
| 124 | 101244.59 |
| 321 | 85559.12 |
| 239 | 80375.24 |
| *** 此处省略了一大波数据 *** |
| 219 | 3452.75 |
| 216 | 3101.4 |
| 161 | 2434.25 |
| 172 | 1960.8 |
+-----+
98 rows in set
```

使用子查询创建视图

以下说明如何使用子查询创建视图，该视图包含价格高于所有产品的平均价格的产品。

```
CREATE VIEW aboveAvgProducts AS
SELECT
    productCode, productName, buyPrice
FROM
    products
WHERE
    buyPrice >
(SELECT
    AVG(buyPrice)
FROM
    products)
ORDER BY buyPrice DESC;
```

查询上述视图：`aboveAvgProducts` 的数据简单如下：

```
SELECT
    *
FROM
    aboveAvgProducts;
```

执行上面查询语句，得到以下结果 -

productCode	productName	buyPrice
S10_4962	1962 Lancia A Delta 16V	103.42
S18_2238	1998 Chrysler Plymouth Prowler	101.51
S10_1949	1952 Alpine Renault 1300	98.58
*****省略了一大波数据*****		
S18_3320	1917 Maxwell Touring Car	57.54
S24_4258	1936 Chrysler Airflow	57.46
S18_3233	1985 Toyota Supra	57.01
S18_2870	1999 Indy 500 Monte Carlo SS	56.76
S32_4485	1974 Ducati 350 Mk3 Desmo	56.13
S12_4473	1957 Chevy Pickup	55.7
S700_3167	F/A 18 Hornet 1/72	54.4

54 rows in set		

在本教程中，我们向您展示了如何使用 CREATE VIEW 语句创建视图。

在本教程中，我们将向您展示如何通过视图创建可更新视图并更新基础表中的数据。

MySQL 可更新视图简介

在MySQL中，视图不仅是可查询的，而且是可更新的。这意味着您可以使用`INSERT`或`UPDATE`语句通过可更新视图插入或更新基表的行。另外，您也可以使用`DELETE`语句通过视图删除底层表的行。

但是，要创建可更新视图，定义视图的`SELECT`语句不能包含以下任何元素：

- 聚合函数，如：`MIN`，`MAX`，`SUM`，`AVG`，`COUNT`等。
- `DISTINCT`子句
- `GROUP BY`子句
- `HAVING`子句
- `UNION`或 `UNION ALL` 子句
- 左连接或外连接。
- `SELECT`子句中的子查询或引用该表的`WHERE`子句中的子查询出现在 `FROM` 子句中。
- 引用 `FROM` 子句中的不可更新视图
- 仅引用文字值
- 对基表的任何列的多次引用

如果使用`TEMPTABLE`算法创建视图，则无法更新视图。

请注意，有时可以使用内部连接创建基于多个表的可更新视图。

MySQL 可更新视图示例

让我们先来看看如何创建一个可更新的视图。

首先，基于示例数据库(yiibaidb)中的`offices`表创建一个名为`officeInfo`的视图。该视图指的是`offices`表中的三列：`officeCode`，`phone`和`city`。

```
CREATE VIEW officeInfo
AS
  SELECT officeCode, phone, city
  FROM offices;
```

接下来，使用以下语句从 `officeInfo` 视图中查询数据：

```
SELECT
*
FROM
officeInfo;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT * FROM officeInfo;
+-----+-----+-----+
| officeCode | phone          | city      |
+-----+-----+-----+
| 1          | +1 650 219 4782 | San Francisco |
| 2          | +1 215 837 0825 | Boston    |
| 3          | +1 212 555 3000 | NYC       |
| 4          | +33 14 723 4404 | Paris     |
| 5          | +86 33 224 5000 | Beijing   |
| 6          | +61 2 9264 2451 | Sydney    |
| 7          | +44 20 7877 2041 | London    |
+-----+-----+-----+
7 rows in set
```

然后，使用以下`UPDATE`语句通过 `officeInfo` 视图更改 `officeCode` 的值为：4 的办公室电话号码。

```
UPDATE officeInfo
SET
phone = '+86 089866668888'
WHERE
officeCode = 4;
```

最后，验证更改结果，通过执行以下查询来查询 `officeInfo` 视图中的数据：

```
mysql> SELECT
    *
  FROM
    officeInfo
 WHERE
    officeCode = 4;

+-----+-----+
| officeCode | phone           | city   |
+-----+-----+
|     4      | +86 089866668888 | Paris |
+-----+-----+
1 row in set
```

检查可更新视图信息

通过从 `information_schema` 数据库中的 `views` 表查询 `is_updatable` 列来检查数据库中的视图是否可更新。

以下查询语句将查询 `yiibaidb` 数据库获取所有视图，并显示哪些视图是可更新的。

```
SELECT
  table_name, is_updatable
FROM
  information_schema.views
WHERE
  table_schema = 'yiibaidb';
```

执行上面查询语句，得到以下结果 -

table_name	is_updatable
aboveavgproducts	YES
bigsalesorder	YES
customerorders	NO
officeinfo	YES
saleperorder	NO

5 rows in set

通过视图删除行

首先，创建一个名为 `items` 的表，在 `items` 表中插入一些行，并创建一个查询包含价格大于 `700` 的项的视图。

```

USE testdb;
-- create a new table named items
CREATE TABLE items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(11 , 2 ) NOT NULL
);

-- insert data into the items table
INSERT INTO items(name,price)
VALUES('Laptop',700.56),('Desktop',699.99),('iPad',700.50) ;

-- create a view based on items table
CREATE VIEW LuxuryItems AS
    SELECT
        *
    FROM
        items
    WHERE
        price > 700;
-- query data from the LuxuryItems view
SELECT
    *
FROM
    LuxuryItems;

```

执行上面查询语句后，得到以下结果 -

id	name	price
1	Laptop	700.56
3	iPad	700.5

2 rows in set

其次，使用 `DELETE` 语句来删除 `id` 为 3 的行。

```
DELETE FROM LuxuryItems  
WHERE  
    id = 3;
```

MySQL返回一条消息，表示有 1 行受到影响。

```
Query OK, 1 row affected
```

第三步，再次通过视图检查数据。

```
mysql> SELECT * FROM LuxuryItems;  
+----+-----+-----+  
| id | name | price |  
+----+-----+-----+  
| 1  | Laptop | 700.56 |  
+----+-----+-----+  
1 row in set
```

第四步，还可以从基表 items 查询数据，以验证 DELETE 语句是否实际删除了该行。

```
mysql> SELECT * FROM items;  
+----+-----+-----+  
| id | name | price |  
+----+-----+-----+  
| 1  | Laptop | 700.56 |  
| 2  | Desktop | 699.99 |  
+----+-----+-----+  
2 rows in set
```

如上面所示，ID 为 3 的行在基表中被删除。

在本教程中，我们向您展示了如何通过创建可更新视图，并更新基础表中的数据。

在本教程中，您将学习如何使用 `WITH CHECK OPTION` 子句确保视图的一致性。

WITH CHECK OPTION子句简介

有时候，[创建](#)一个视图来显示表的部分数据。然而，简单视图是[可更新的](#)，因此可以更新通过视图不可见的数据。此更新使视图不一致。为了确保视图的一致性，在创建或修改视图时使用 `WITH CHECK OPTION` 子句。

下面说明了 `WITH CHECK OPTION` 子句的语法 -

```
CREATE OR REPLACE VIEW view_name  
AS  
    select_statement  
    WITH CHECK OPTION;
```

请注意，将分号(;)放在 `WITH CHECK OPTION` 子句的末尾，而不是在[SELECT](#)语句的末尾来定义视图。

我们来看一下使用 `WITH CHECK OPTION` 子句的例子。

MySQL WITH CHECK OPTION子句示例

首先，我们根据 `employees` 表[创建](#)一个名为 `vps` 的视图，以显示其职位为 `VP` 的员工，例如 `VP Marketing` 和 `VP Sales` 。

```
CREATE OR REPLACE VIEW vps AS
SELECT
    employeeNumber,
    lastname,
    firstname,
    jobtitle,
    extension,
    email,
    officeCode,
    reportsTo
FROM
    employees
WHERE
    jobTitle LIKE '%VP%';
```

接下来，使用以下语句从 `vps` 视图中查询数据：

```
SELECT * FROM vps;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT * FROM vps;
+-----+-----+-----+-----+-----+
| employeeNumber | lastname | firstname | jobtitle      | extensi
on | email           | officeCode | reportsTo |
+-----+-----+-----+-----+-----+
|     1056 | Hill     | Mary       | VP Sales      | x4611
| mary.hill@yiibai.com | 1          |             | 1002         |
|     1076 | Firrelli  | Jeff       | VP Marketing   | x9273
| jfirrelli@yiibai.com | 1          |             | 1002         |
+-----+-----+-----+-----+-----+
2 rows in set
```

因为 `vps` 是一个简单的视图，因此它是可更新的。

然后，我们通过 `vps` 视图将一行员工数据信息插入。

```
INSERT INTO vps(employeeNumber,firstname,lastname,jobtitle,extension,email,officeCode,reportsTo)
values(1703,'Lily','Bush','IT Manager','x9111','lilybush@yiiibai.com',1,1002);
```

请注意，新创建的员工通过 `vps` 视图不可见，因为她的职位是 `IT` 经理，而不是 `VP`。使用以下 `SELECT` 语句来验证它。

```
SELECT * FROM employees WHERE employeeNumber=1703;
```

执行上面语句，得到以下结果 -

employeeNumber	lastName	firstName	extension	email
officeCode	reportsTo	jobTitle		
1703	Bush	Lily	x9111	lilybush@yiiibai.com
1		1002	IT Manager	
1702	Gerard	Martin	x2312	mgerard@gmail.com
4		1102	Sales Rep	
1625	Kato	Yoshimi	x102	ykato@gmaiil.com
5		1621	Sales Rep	
1621	Nishi	Mami	x101	mnishi@gmaiail.com
5		1056	Sales Rep	

但这可能不是我们想要的，因为通过 `vps` 视图暴露 `VP` 员工，而不是其他员工。

为了确保视图的一致性，用户只能显示或更新通过视图可见的数据，则在创建或修改视图时使用 `WITH CHECK OPTION`。

让我们修改视图以包括 `WITH CHECK OPTION` 选项。

```
CREATE OR REPLACE VIEW vps AS
SELECT
    employeeNumber,
    lastname,
    firstname,
    jobtitle,
    extension,
    email,
    officeCode,
    reportsTo
FROM
    employees
WHERE
    jobTitle LIKE '%VP%'
WITH CHECK OPTION;
```

请注意在 CREATE OR REPLACE 语句的结尾处加上 WITH CHECK OPTION 子句。

之后，再次通过 vps 视图将一行插入 employees 表中，如下所示：

```
INSERT INTO vps(employeeNumber,firstname,lastname,jobtitle,extension,email,officeCode,reportsTo)
VALUES(1704,'John','Minsu','IT Staff','x9112','johnminsu@yiibai.com',1,1703);
```

这次MySQL拒绝插入并发出以下错误消息：

```
Error Code: 1369 - CHECK OPTION failed 'yiibaidb.vps'
```

最后，我们通过 vps 视图将一个职位为 SVP Marketing 的员工插入 employees 表，看看MySQL是否允许这样做。

```
INSERT INTO vps(employeeNumber,firstname,lastname,jobtitle,extension,email,officeCode,reportsTo)
VALUES(1704,'John','Minsu','SVP Marketing','x9112','johnminsu@classicmodelcars.com',1,1076);
```

MySQL发出 1 行受影响(Query OK, 1 row affected)。

可以通过根据 `vps` 视图查询数据来再次验证插入操作。

```
SELECT * FROM vps;
```

如上查询结果所示，它的确按预期工作了。

```
mysql> SELECT * FROM vps;
+-----+-----+-----+-----+-----+
| employeeNumber | lastname | firstname | jobtitle      | extensio
n | email
+-----+-----+-----+-----+-----+
| 1056 | Hill    | Mary     | VP Sales      | x4611
| mary.hill@yiibai.com |           | 1            | 1002
| 1076 | Firrelli | Jeff     | VP Marketing   | x9273
| jfirrelli@yiibai.com |           | 1            | 1002
| 1704 | Minsu    | John     | SVP Marketing  | x9112
| johnminsu@classicmodelcars.com | 1           |               | 1076
+-----+-----+-----+-----+-----+
3 rows in set
```

在本教程中，您学习了如何使用 `WITH CHECK OPTION` 子句来确保视图的一致性。

本教程通过示例和清楚的说明帮助，了解 `WITH CHECK OPTION` 子句中 `LOCAL` 和 `CASCDED` 之间的差异。

在进行本教程之前，应该熟悉 `WITH CHECK OPTION` 子句。如果不是这样，可以参阅[WITH CHECK OPTION子句教程](#)来遵循确保视图的一致性。

LOCAL&CASCDED检查范围介绍

当使用 `WITH CHECK OPTION` 子句[创建视图](#)时，MySQL会通过视图检查正在更改的每个行，例如[插入](#)，[更新](#)，[删除](#)，以使其符合视图的定义。因为MySQL允许基于另一个视图创建视图，它还会检查依赖视图中的规则以保持一致性。

为了确定检查的范围，MySQL提供了两个选项：`LOCAL` 和 `CASCDED`。如果您没有在 `WITH CHECK OPTION` 子句中显式指定关键字，则MySQL默认使用 `CASCDED`。

MySQL与CASCADC检查选项

要了解使用 `CASCDED CHECK OPTION` 的效果，请参阅下面的例子。

首先，[创建一个名为 `t1` 的表](#)，其中只有一个名称为：`c` 的列，它的数据类型为 `int`。

```
USE testdb;
CREATE TABLE t1 (
    c INT
);
```

接下来，基于 `t1` 表创建一个名为 `v1` 的视图，以选择值大于 `10` 的行记录。

```
CREATE OR REPLACE VIEW v1
AS
SELECT
    c
FROM
    t1
WHERE
    c > 10;
```

因为没有指定 `WITH CHECK OPTION`，所以以下语句即使不符合 `v1` 视图的定义也可以工作。

```
INSERT INTO v1(c) VALUES (5);
```

然后，基于 `v1` 视图创建 `v2` 视图。在 `v2` 视图中添加一个 `WITH CASCDED CHECK OPTION` 子句。

```
CREATE OR REPLACE VIEW v2
AS
SELECT
    c
FROM
    v1
WITH CASCDED CHECK OPTION;
```

现在，通过 `v2` 视图在 `t1` 表中插入一个值为 5 的行。

```
INSERT INTO v2(c) VALUES (5);
```

MySQL发出以下错误消息：

```
Error Code: 1369. CHECK OPTION failed 'testdb.v2'
```

它失败了，因为它创建一个不符合 `v2` 视图定义的新行。

之后，我们再创建一个基于 `v2` 的名为 `v3` 的新视图。

```
CREATE OR REPLACE VIEW v3
AS
SELECT
    c
FROM
    v2
WHERE
    c < 20;
```

我们通过 `v3` 视图插入一个新行到 `t1` 表中，值为 `8`。

```
INSERT INTO v3(c) VALUES (8);
```

MySQL发出以下错误信息：

```
Error Code: 1369. CHECK OPTION failed 'testdb.v3'
```

上面插入语句看起来符合 `v3` 视图的定义，`insert`语句仍然执行失败。

这是为什么呢？

因为 `v3` 视图取决于 `v2` 视图，`v2` 视图具有 `WITH CASCADED CHECK OPTION`。

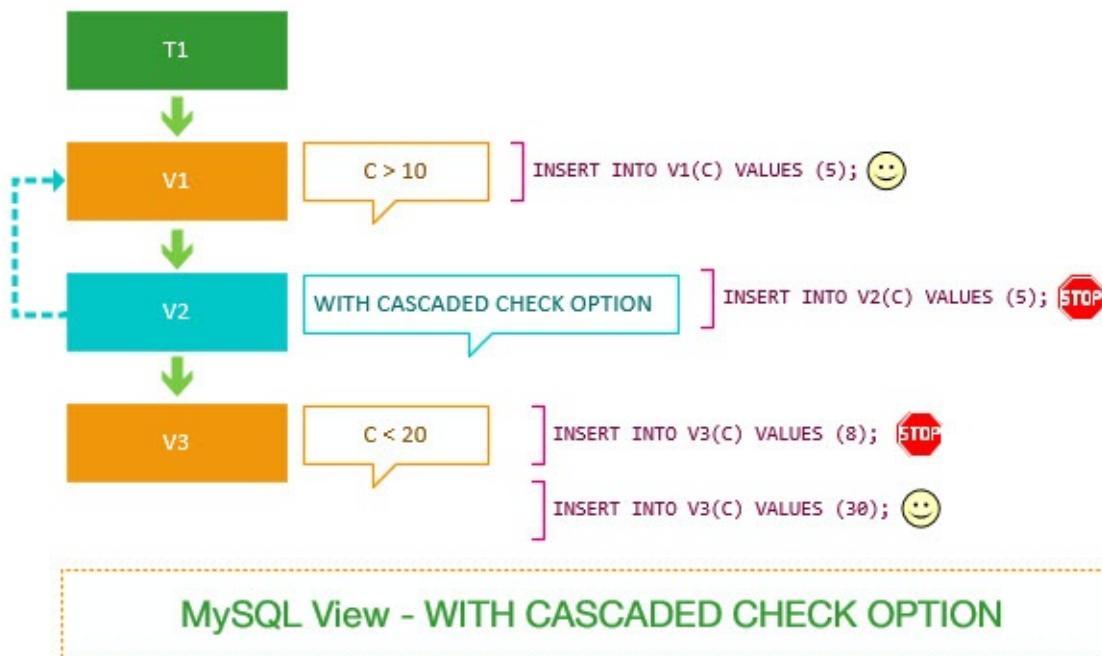
但是，以下插入语句能正常工作。

```
INSERT INTO v3(c) VALUES (30);
```

因为 `v3` 视图没有使用 `WITH CHECK OPTION` 定义，并且该语句符合 `v2` 视图的定义。

所以，总而言之：

当视图使用 `WITH CASCADED CHECK OPTION` 时，MySQL会循环检查视图的规则以及底层视图的规则。



MySQL WITH LOCAL CHECK OPTION

下面将演示使用 `WITH LOCAL CHECK OPTION` 选项，使用上面相同的示例来查看差异。

首先，将 `v2` 视图更改为使用 `WITH LOCAL CHECK OPTIONS` 替代。

```
ALTER VIEW v2 AS
  SELECT
    c
  FROM
    v1
  WITH LOCAL CHECK OPTION;
```

其次，插入与上述示例相同的行。

```
INSERT INTO v2(c) VALUES (5);
```

它是可以成功执行的。

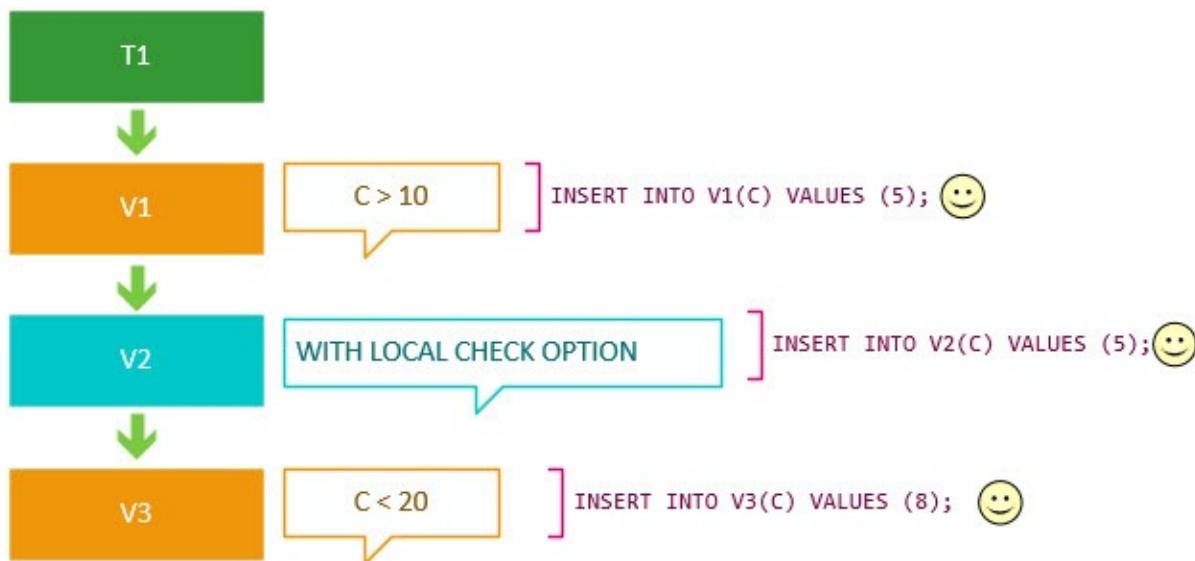
因为 `v2` 视图没有任何规则。 `v2` 视图取决于 `v1` 视图。但是，`v1` 视图没有指定检查选项，因此MySQL跳过检查 `v1` 视图中的规则。

请注意，在使用 `WITH CASCADED CHECK OPTION` 创建的 `v2` 视图中，此语句失败。

第三，通过 `v3` 视图将相同的行插入 `t1` 表。

```
INSERT INTO v3(c) VALUES (8);
```

在这种情况下可以执行成功，因为MySQL视图中的 `WITH LOCAL CHECK OPTIONS` 选项没有检查 `v1` 视图的规则。另外，请注意，在使用 `WITH CASCADED CHECK OPTION` 创建的 `v2` 视图示例中，此语句执行失败。



MySQL View - WITH LOCAL CHECK OPTION

因此，如果视图使用 `WITH LOCAL CHECK OPTION`，MySQL会检查 `WITH LOCAL CHECK OPTION` 和 `WITH CASCADED CHECK OPTION` 选项的视图规则。

与使用 `WITH CASCADED CHECK OPTION` 的视图不同，MySQL检查所有依赖视图的规则。

请注意，在MySQL 5.7.6之前，如果您使用带有 `WITH LOCAL CHECK OPTION` 的视图，MySQL只会检查当前视图的规则，并且不会检查底层视图的规则。

在本教程中，您将学习如何管理MySQL中的视图，包括显示，修改和删除视图。

查看视图定义

MySQL提供了用于显示视图定义的 `SHOW CREATE VIEW` 语句。

以下是 `SHOW CREATE VIEW` 语句的语法：

```
SHOW CREATE VIEW [database_name].[view_name];
```

要显示视图的定义，需要在 `SHOW CREATE VIEW` 子句之后指定视图的名称。

为了更好的演示，我们先来[创建一个视图](#)。

假设根据 `employees` 表创建一个简单的视图用来显示公司组织结构：

```
USE yiibaidb;
CREATE VIEW organization AS
    SELECT
        CONCAT(E.lastname, E.firstname) AS Employee,
        CONCAT(M.lastname, M.firstname) AS Manager
    FROM
        employees AS E
        INNER JOIN
        employees AS M ON M.employeeNumber = E.ReportsTo
    ORDER BY Manager;
```

从以上视图中查询数据，得到以下结果 -

```
mysql> SELECT * FROM organization;
+-----+-----+
| Employee | Manager |
+-----+-----+
| BondurLou | BondurGerard |
| CastilloPamela | BondurGerard |
| JonesBarry | BondurGerard |
| HernandezGerard | BondurGerard |
| .....此处省略了一大波数据..... |
| KatoYoshimi | NishiMami |
| KingTom | PattersonWilliam |
| MarshPeter | PattersonWilliam |
| FixterAndy | PattersonWilliam |
+-----+-----+
24 rows in set
```

要显示视图的定义，请使用 `SHOW CREATE VIEW` 语句如下：

```
SHOW CREATE VIEW organization;
```

还可以使用任何纯文本编辑器(如记事本)显示视图的定义，以打开数据库文件夹中的视图定义文件。

例如，要打开 `organization` 视图定义，可以使用以下路径找到视图定义文件：`\data\yiibaidb\organization.frm`。

但是，不应该直接在 `.frm` 文件中修改视图的定义。

修改视图

MySQL提供两个语句，允许您修改现有视图：`ALTER VIEW` 和 `CREATE OR REPLACE VIEW`。

使用**`ALTER VIEW`**语句修改视图

创建视图后，可以使用 `ALTER VIEW` 语句修改视图。

`ALTER VIEW` 语句的语法类似于 `CREATE VIEW` 语句，除了 `CREATE` 关键字被 `ALTER` 关键字替换外，其它都一样。

```
ALTER
[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]
VIEW [database_name]. [view_name]
AS
[SELECT statement]
```

以下语句通过添加 `email` 列来演示如何修改 `organization` 视图。

```
ALTER VIEW organization
AS
SELECT CONCAT(E.lastname, E.firstname) AS Employee,
       E.email AS employeeEmail,
       CONCAT(M.lastname, M.firstname) AS Manager
FROM employees AS E
INNER JOIN employees AS M
ON M.employeeNumber = E.ReportsTo
ORDER BY Manager;
```

要验证更改，可以从 `organization` 视图中查询数据：

```
SELECT
*
FROM
Organization;
```

执行上面查询语句，得到以下结果 -

使用**CREATE OR REPLACE VIEW**语句修改视图

除 `ALTER VIEW` 语句外，还可以使用 `CREATE OR REPLACE VIEW` 语句来创建或替换现有视图。如果一个视图已经存在，MySQL 只会修改视图。如果视图不存在，MySQL 将创建一个新的视图。

以下语句使用 `CREATE OR REPLACE VIEW` 语法根据 `employees` 表创建一个名称为 `v_contacts` 的视图：

```
CREATE OR REPLACE VIEW v_contacts AS
  SELECT
    firstName, lastName, extension, email
  FROM
    employees;
-- 查询视图数据
SELECT * FROM v_contacts;
```

执行上面查询语句，得到以下结果 -

firstName	lastName	extension	email
Diane	Murphy	x5800	dmurphy@yiibai.com
Mary	Hill	x4611	mary.hill@yiibai.com
Jeff	Firrelli	x9273	jfirrelli@yiibai.com
William	Patterson	x4871	wpatterson@yiibai.com
Gerard	Bondur	x5408	gbondur@gmail.com
Anthony	Bow	x5428	abow@gmail.com
Leslie	Jennings	x3291	ljennings@yiibai.com
此处省略了一大波数据			
Martin	Gerard	x2312	mgerard@gmail.com
Lily	Bush	x9111	lilybush@yiibai.com
John	Minsu	x9112	johnminsu@classicmodelcars.com

25 rows in set

假设您要将职位(jobtitle)列添加到 v_contacts 视图中，只需使用以下语句 -

```

CREATE OR REPLACE VIEW v_contacts AS
SELECT
    firstName, lastName, extension, email, jobtitle
FROM
    employees;
-- 查询视图数据
SELECT * FROM v_contacts;

```

执行上面查询语句后，可以看到添加一列数据 -

firstName	lastName	extension	email
jobtitle			
Diane	Murphy	x5800	dmurphy@yiibai.com
	President		
Mary	Hill	x4611	mary.hill@yiibai.com
	VP Sales		
Jeff	Firrelli	x9273	jfirrelli@yiibai.com
	VP Marketing		
此处省略了一大波数据			
Yoshimi	Kato	x102	ykato@gmail.com
	Sales Rep		
Martin	Gerard	x2312	mgerard@gmail.com
	Sales Rep		
Lily	Bush	x9111	lilybush@yiibai.com
	IT Manager		
John	Minsu	x9112	johnminsu@classicmodelcars
.com	SVP Marketing		
25 rows in set			

删除视图

创建视图后，可以使用 `DROP VIEW` 语句将其删除。下面说明了 `DROP VIEW` 语句的语法：

```
DROP VIEW [IF EXISTS] [database_name].[view_name]
```

`IF EXISTS` 是语句的可选子句，它允许您检查视图是否存在。它可以避免删除不存在的视图的错误。

例如，如果要删除 `organization` 视图，可以按如下所示使用 `DROP VIEW` 语句：

```
DROP VIEW IF EXISTS organization;
```

每次修改或删除视图时，MySQL会将视图定义文件备份到 `/database_name/arc/` 目录中。如果您意外修改或删除视图，可以从 `/database_name/arc/` 文件夹获取其备份。

在本教程中，您已经学会了如何管理MySQL中的视图，包括显示，修改和删除视图。

第六章 全文搜索

在本节中，您将学习如何使用MySQL全文搜索功能。MySQL全文搜索提供了一种实现各种高级搜索技术的简单方法，如自然语言搜索，布尔文本搜索和查询扩展。

1. MySQL全文搜索简介

- 简要介绍MySQL全文搜索及其功能。参考阅读：
读：<http://www.yiibai.com/mysql/introduction-to-mysql-full-text-search.html>

2. 定义MySQL全文搜索的**FULLTEXT**索引

- 在本教程中，您将学习如何定义数据库表中列的全文索引，然后才能执行全文搜索。参考阅读：<http://www.yiibai.com/mysql/activating-full-text-searching.html>

3. MySQL自然语言全文搜索

- 基本上在自然语言搜索中，MySQL查找与自由文本自然人类语言查询相关的行或文档，例如“如何使用MySQL全文搜索”。参考阅读：
读：<http://www.yiibai.com/mysql/natural-language-search.html>

4. MySQL布尔全文搜索

- 您将了解MySQL布尔全文搜索及其主要功能。我们使用一些例子讲解，帮助您更好地理解这个概念。参考阅读：<http://www.yiibai.com/mysql/boolean-text-searches.html>

5. 使用MySQL查询扩展

- 我们向您展示了MySQL全文搜索的一个非常重要的功能，称为查询扩展。参考阅读：<http://www.yiibai.com/mysql/using-mysql-query-expansion.html>

6. MySQL ngram全文解析器

- 本教程将向您展示如何使用MySQL ngram全文解析器来支持中文，日文，韩文等表意语言的全文搜索。参考阅读：<http://www.yiibai.com/mysql/ngram-full-text-parser.html>

在本教程中，我们将向您介绍MySQL全文搜索及其功能。

MySQL支持使用**LIKE**运算符和**正则表达式**进行文本搜索。但是，当文本列较大并且表中的行数增加时，使用这些方法有一些限制：

- **性能问题**：MySQL必须扫描整个表以根据正则表达式中的 `LIKE` 语句或模式中的模式查找确切的文本。
- **灵活搜索**：使用 `LIKE` 运算符和正则表达式搜索，很难进行灵活的搜索查询，例如，查找描述包含 `car` 但不是 `classic` 的产品。
- **相关性排名**：没有办法指定结果集中的哪一行与搜索字词更相关。

由于这些限制，MySQL扩展了一个非常好的功能，叫作全文搜索。从技术上讲，MySQL从启用的全文搜索列的单词中创建一个索引，并对该索引进行搜索。

MySQL使用复杂的算法来确定与搜索查询匹配的行。

以下是MySQL全文搜索的一些重要功能：

- **本地SQL类接口**：使用类似SQL的语句来使用全文搜索。
- **完全动态的索引**：当该列的数据发生变化时，MySQL会自动更新文本列的索引。
- **适度的索引大小**：它不需要太多的内存来存储索引。
- **最后一个是，基于复杂的搜索查询快速搜索。**

请注意，并非所有**存储引擎**都支持全文搜索功能。在MySQL 5.6或更高版本中，只有**MyISAM**和**InnoDB**存储引擎支持全文搜索。

在本教程中，您将学习如何定义全文索引，以便在MySQL中执行各种全文搜索。

在表的列中执行全文搜索之前，必须对其数据进行索引。每当列的数据更改时，MySQL将重新创建全文索引。在MySQL中，全文索引是一种名称为 `FULLTEXT` 的索引。

MySQL支持对全文搜索启用列自动建立索引和重新索引数据。MySQL 5.6或更高版本允许您为数据类型为 `MyISAM` 中的 `CHAR`，`VARCHAR` 或 `TEXT` 或 `InnoDB` 表类型的列定义全文索引。请注意，自MySQL5.6版以来，MySQL支持 `InnoDB` 表的全文索引。

MySQL允许您在使用`CREATE TABLE`语句创建表或为现有表使用`ALTER TABLE`或`CREATE INDEX`语句时来定义 `FULLTEXT` 索引。

使用`CREATE TABLE`语句定义`FULLTEXT`索引

通常，使用 `CREATE TABLE` 语句创建新表时，可以为列定义 `FULLTEXT` 索引，如下所示：

```
CREATE TABLE table_name(
    column1 data_type,
    column2 data_type,
    column3 data_type,
    ...
    PRIMARY_KEY(key_column),
    FULLTEXT (column1, column2, ...)
);
```

要创建 `FULLTEXT` 索引，请在 `FULLTEXT` 关键字之后的括号中放置逗号分隔列的列表。

以下语句创建一个名为 `posts` 的新表，该表具有包含 `post_content` 列的 `FULLTEXT` 紴引。

```
CREATE TABLE posts (
    id int(4) NOT NULL AUTO_INCREMENT,
    title varchar(255) NOT NULL,
    post_content text,
    PRIMARY KEY (id),
    FULLTEXT KEY post_content (post_content)
);
```

定义现有表的**FULLTEXT**索引

如果您想要在一个已存在表上定义全文索引，可以使用 `ALTER TABLE` 语句或 `CREATE INDEX` 语句。

使用 `ALTER TABLE` 语句定义 **FULLTEXT** 索引

以下语法使用 `ALTER TABLE` 语句定义 `FULLTEXT` 索引：

```
ALTER TABLE table_name
ADD FULLTEXT(column_name1, column_name2, ...)
```

将 `table_name` 指定为一列或多列定义 `FULLTEXT` 索引的 `ADD FULLTEXT` 子句。

例如，可以在 [示例数据库\(yiibaidb\)](#) 的 `products` 表中为 `productDescription` 和 `productLine` 列定义 `FULLTEXT` 紴引，如下所示：

```
ALTER TABLE products
ADD FULLTEXT(productDescription, productLine)
```

使用 `CREATE INDEX` 语句定义 **FULLTEXT** 索引

还可以使用 `CREATE INDEX` 语句为现有表创建 `FULLTEXT` 紴引。请参阅以下语法：

```
CREATE FULLTEXT INDEX index_name
ON table_name(idx_column_name, ...)
```

以下语句为 `offices` 表的 `addressLine1` 和 `addressLine2` 列创建一个 `FULLTEXT` 索引。

```
CREATE FULLTEXT INDEX address  
ON offices(addressLine1, addressLine2)
```

请注意，对于具有多行的表，首先将数据加载到没有 `FULLTEXT` 索引的表中，然后创建 `FULLTEXT` 索引，而不是将大量数据加载到具有现有 `FULLTEXT` 索引的表中，先加载数据后创建索引速度更快。

删除全文搜索列

要删除 `FULLTEXT` 索引，只需使用 `ALTER TABLE ... DROP INDEX` 语句删除索引。例如，以下语句删除了 `offices` 表中名称为 `address` 的 `FULLTEXT` 紴引，可使用以下语句 -

```
ALTER TABLE offices  
DROP INDEX address;
```

在本教程中，我们向您展示了如何定义和删除支持MySQL全文搜索的 `FULLTEXT` 紴引。

在本教程中，您将通过使用 `MATCH()` 和 `AGAINST()` 函数来了解MySQL自然语言全文搜索。

MySQL自然语言全文搜索简介

在自然语言全文搜索中，MySQL查找与自由文本自然人类语言查询相关的行或文档，例如“如何使用MySQL自然语言全文搜索”。

相关性是一个正浮点数。当相关性为零时，这意味着没有相似性。MySQL根据各种因素计算相关性，包括文档中的字数，文档中的唯一字数，集合中的单词总数以及包含特定单词的文档数(行)。

要执行自然语言全文搜索，您可以使用 `MATCH()` 和 `AGAINST()` 函数。

`MATCH()` 函数指定要搜索的列，`AGAINST()` 函数确定要使用的搜索表达式。

MySQL自然语言全文搜索示例

我们将使用示例数据库([yiibaidb](#))中的 `products` 表进行演示。

```
mysql> desc products;
+-----+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default | Ex |
+-----+-----+-----+-----+-----+
| productCode    | varchar(15) | NO   | PRI |          |     |
| productName    | varchar(70)  | NO   |      | NULL    |     |
| productLine    | varchar(50)  | NO   | MUL |          |     |
| productScale   | varchar(10)  | NO   |      | NULL    |     |
| productVendor  | varchar(50)  | NO   |      | NULL    |     |
| productDescription | text      | NO   |      | NULL    |     |
| quantityInStock | smallint(6) | NO   |      | NULL    |     |
| buyPrice        | decimal(10,2) | NO   |      | NULL    |     |
| MSRP            | decimal(10,2) | NO   |      | NULL    |     |
| stockValue      | double     | YES  |      | NULL    | ST |
+-----+-----+-----+-----+-----+
ORED GENERATED
+-----+-----+-----+-----+-----+
10 rows in set (0.22 sec)
```

首先，需要使用 `ALTER TABLE ADD FULLTEXT` 语句在 `products` 表的 `productLine` 列中启用全文搜索：

```
ALTER TABLE products
ADD FULLTEXT(productline);
```

其次，可以搜索产品系列包含 `Classic` 的产品，使用 `MATCH()` 和 `AGAINST()` 函数，如下查询：

```
SELECT productName, productline
FROM products
WHERE MATCH(productline) AGAINST('Classic');
```

执行上面查询语句，得到以下结果 -

productName	productline
1952 Alpine Renault 1300	Classic Cars
1972 Alfa Romeo GTA	Classic Cars
1962 Lancia A Delta 16V	Classic Cars
1968 Ford Mustang	Classic Cars
2001 Ferrari Enzo	Classic Cars
1969 Corvair Monza	Classic Cars
1968 Dodge Charger	Classic Cars
1969 Ford Falcon	Classic Cars
1970 Plymouth Hemi Cuda	Classic Cars
1969 Dodge Charger	Classic Cars
1993 Mazda RX 7	Classic Cars
1965 Aston Martin DB5	Classic Cars
1948 Porsche 356 A Roadster	Classic Cars
1995 Honda Civic	Classic Cars
1998 Chrysler Plymouth Prowler	Classic Cars
1999 Indy 500 Monte Carlo SS	Classic Cars
1992 Ferrari 360 Spider red	Classic Cars
1985 Toyota Supra	Classic Cars
1969 Dodge Super Bee	Classic Cars
1976 Ford Gran Torino	Classic Cars
1948 Porsche Type 356 Roadster	Classic Cars
1970 Triumph Spitfire	Classic Cars
1957 Corvette Convertible	Classic Cars
1957 Ford Thunderbird	Classic Cars
1970 Chevy Chevelle SS 454	Classic Cars
1970 Dodge Coronet	Classic Cars
1966 Shelby Cobra 427 S C	Classic Cars
1949 Jaguar XK 120	Classic Cars

1958 Chevy Corvette Limited Edition	Classic Cars
1952 Citroen 15CV	Classic Cars
1982 Lamborghini Diablo	Classic Cars
1969 Chevrolet Camaro Z28	Classic Cars
1971 Alpine Renault 1600s	Classic Cars
2002 Chevy Corvette	Classic Cars
1956 Porsche 356A Coupe	Classic Cars
1992 Porsche Cayenne Turbo Silver	Classic Cars
1961 Chevrolet Impala	Classic Cars
1982 Camaro Z28	Classic Cars

38 rows in set

AGAINST() 函数默认使用 IN NATURAL LANGUAGE MODE 搜索修饰符，因此您可以在查询中省略它。还有其他搜索修饰符，例如 IN BOOLEAN MODE 用于布尔文本搜索。

可以在查询中显式使用 IN NATURAL LANGUAGE MODE 搜索修饰符，如下所示：

```
SELECT productName, productline
FROM products
WHERE MATCH(productline)
AGAINST('Classic,Vintage' IN NATURAL LANGUAGE MODE);
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT productName, productline
    FROM products
   WHERE MATCH(productline)
AGAINTS('Classic,Vintage' IN NATURAL LANGUAGE MODE);
+-----+-----+
| productName | productline |
+-----+-----+
| 1937 Lincoln Berline | Vintage Cars |
| 1936 Mercedes-Benz 500K Special Roadster | Vintage Cars |
| 1917 Grand Touring Sedan | Vintage Cars |
| 1911 Ford Town Car | Vintage Cars |
***** 此处省略了一大波数据 ****
| 1971 Alpine Renault 1600s | Classic Cars |
| 2002 Chevy Corvette | Classic Cars |
| 1956 Porsche 356A Coupe | Classic Cars |
| 1992 Porsche Cayenne Turbo Silver | Classic Cars |
| 1961 Chevrolet Impala | Classic Cars |
| 1982 Camaro Z28 | Classic Cars |
+-----+-----+
62 rows in set
```

默认情况下，MySQL以不区分大小写的方式执行搜索。但是，您可以指示MySQL使用二进制排序规则对索引列进行区分大小写搜索。

按相关性排序结果集

全文搜索的一个非常重要的特征是MySQL根据其相关性对结果集中的行进行排序。当WHERE子句中使用 MATCH() 函数时，MySQL返回首先更相关的行。

以下示例显示了MySQL如何根据相关性对结果集进行排序。

首先，可以为 products 表的 productName 列启用全文搜索功能。

```
ALTER TABLE products
ADD FULLTEXT(productName);
```

其次，使用以下查询搜索名称包 Ford 和/或 1932 的产品：

```
SELECT productName, productline
FROM products
WHERE MATCH(productName) AGAINST('1932,Ford');
```

我们来查看输出结果：

```
mysql> SELECT productName, productline
FROM products
WHERE MATCH(productName) AGAINST('1932,Ford');
+-----+-----+
| productName | productline |
+-----+-----+
| 1932 Model A Ford J-Coupe | Vintage Cars |
| 1932 Alfa Romeo 8C2300 Spider Sport | Vintage Cars |
| 1968 Ford Mustang | Classic Cars |
| 1969 Ford Falcon | Classic Cars |
| 1940 Ford Pickup Truck | Trucks and Buses |
| 1911 Ford Town Car | Vintage Cars |
| 1926 Ford Fire Engine | Trucks and Buses |
| 1913 Ford Model T Speedster | Vintage Cars |
| 1934 Ford V8 Coupe | Vintage Cars |
| 1903 Ford Model A | Vintage Cars |
| 1976 Ford Gran Torino | Classic Cars |
| 1940s Ford truck | Trucks and Buses |
| 1957 Ford Thunderbird | Classic Cars |
| 1912 Ford Model T Delivery Wagon | Vintage Cars |
| 1940 Ford Delivery Sedan | Vintage Cars |
| 1928 Ford Phaeton Deluxe | Vintage Cars |
+-----+-----+
16 rows in set
```

首先返回其名称包含 1932 和 Ford 的产品，然后返回名称包含唯一 Ford 关键字的产品。

使用全文搜索时，应该记住一些重点：

- MySQL全文搜索引擎中定义的搜索项的最小长度为 4，这意味着如果搜索长度小于 4 的关键字，例如 car，cat 等，则不会得到任何结果。
- 停止词被忽略，MySQL定义了MySQL源代码分

发 `storage/myisam/ft_static.c` 中的停止词列表。

在本教程中，向您展示了如何使用 `MATCH()` 和 `AGAINST()` 函数在MySQL中执行自然语言搜索。

在本教程中，您将学习如何执行MySQL布尔全文搜索。此外，您将学习如何使用布尔运算符来组成非常复杂的搜索查询。

MySQL简介全文检索

除了[自然语言全文搜索](#)，MySQL还支持一种叫作布尔全文搜索的全文搜索的附加形式。在布尔模式中，MySQL搜索词而不是自然语言搜索中的概念。

MySQL允许您根据布尔模式下的非常复杂的查询以及布尔运算符执行全文搜索。这就是为什么布尔模式的全文搜索适合有经验的用户。

要在布尔模式下执行全文搜索，您可以在 `AGAINST` 表达式中使用 `IN BOOLEAN MODE` 修饰符。以下示例说明如何搜索产品名称中包含单词 `Truck` 的产品。

```
SELECT productName, productline
FROM products
WHERE MATCH(productName)
      AGAINST('Truck' IN BOOLEAN MODE )
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT productName, productline
    FROM products
   WHERE MATCH(productName)
         AGAINST('Truck' IN BOOLEAN MODE );
+-----+-----+
| productName | productline |
+-----+-----+
| 1940 Ford Pickup Truck | Trucks and Buses |
| 1940s Ford truck | Trucks and Buses |
+-----+
2 rows in set
```

返回产品名称包含 `Truck` 的两个产品。

要查找产品名称包含单词 `Truck` 但不包含 `Pickup` 的行的产品，可以使用排除布尔运算符(`-`)，该运算符返回不包括 `Pickup` 关键字的结果，如以下查询：

```
SELECT productName, productline
FROM products
WHERE MATCH(productName) AGAINST('Truck -Pickup' IN BOOLEAN MODE
);
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT productName, productline
FROM products
WHERE MATCH(productName) AGAINST('Truck -Pickup' IN BOOLEAN MODE
);
+-----+-----+
| productName | productline |
+-----+-----+
| 1940s Ford truck | Trucks and Buses |
+-----+
1 row in set
```

MySQL布尔全文搜索运算符

下表说明了全文搜索布尔运算符及其含义：

操作符	描述
+	包括，这个词必须存在。
-	排除，这个词不能存在。
>	包括并增加排名值。
<	包括并降低排名值。
()	将单词分组成子表达式(允许将其包括，排除，排序等作为一个组)。
~	否定一个词的排名值。
*	通配符，在结尾的单词
""	定义一个短语(与单个单词列表相反，整个短语匹配包含或排除)。

以下示例说明如何在搜索查询中使用布尔全文运算符：

要搜索包含两个词中至少一个词的行：`mysql` 或 `tutorial`，可使用：`mysql tutorial`；要搜索包含两个单词的行：`mysql` 或 `tutorial`，可使用：`+mysql +tutorial`；要搜索包含单词“`mysql`”的行，但为含有“`tutorial`”的行排列较高的排名：可使用：`+mysql tutorial`；要搜索包含单词“`mysql`”而不是“`tutorial`”的行，可使用：`+mysql -tutorial`；要搜索包含单词“`mysql`”的行，如果包含单词“`tutorial`”，则将行排列，可使用：`+mysql ~tutorial` 要搜索包含单词“`mysql`”和“`tutorial`”，或“`mysql`”和“`training`”的行以任何顺序排列，但将包含“`mysql tutorial`”的行高于“`mysql training`”。可使用：`+mysql +(> tutorial < training)`

要查找包含以“`my`”开头的单词的行，例如“`mysql`”，“`myspace`”等，请使用以下命令：`my*`。

MySQL布尔全文搜索主要功能

- MySQL不按照布尔全文搜索中相关性降低的顺序自动排序行。
- 要执行布尔查询，*InnoDB*表需要*MATCH*表达式的所有列具有 `FULLTEXT` 索引。请注意，*MyISAM*表不需要这个，尽管搜索速度相当慢。
- MySQL在*InnoDB*表上的搜索查询上不支持多个布尔运算符，例如 `++mysql`。如果这样做，MySQL将返回错误。但是，*MyISAM*的行为方式不一样。它忽略其他运算符并使用最接近搜索词的运算符，例如`+ -mysql`将成为 `-mysql`。
- *InnoDB*全文搜索不支持尾部加号(+)或减号(-)号。它只支持前加号或减号。如果搜索字是 `mysql+` 或 `mysql-`，MySQL将会报错。另外，以下带有通配符的正加号或负号无效：`+*`，`+-`
- 不适用 `50%` 阈值。顺便说一下，`50%` 阈值意味着如果一个词出现在超过 `50%` 的行中，MySQL将在搜索结果中忽略它。

在本教程中，我们向您展示了如何使用许多有用的布尔运算符执行MySQL布尔全文搜索。

在本教程中，您将了解到基于自动相关性反馈的MySQL查询扩展扩展搜索结果。

MySQL查询扩展简介

在某些情况下，用户希望根据他们拥有的知识来搜索信息。用户使用他们的知识来定义关键字来搜索信息，通常这些关键字太短。

为了帮助用户根据很短的关键字找到他们想要的内容，MySQL全文搜索引擎引入了一个称为查询扩展的概念。

查询扩展用于根据自动相关性反馈(或盲查询扩展)来扩大全文搜索的搜索结果。从技术上讲，当使用查询扩展时，MySQL全文搜索引擎将执行以下步骤：

- 首先，MySQL全文搜索引擎会查找与搜索查询匹配的所有行。
- 其次，它检查搜索结果中的所有行，并找到相关词。
- 第三，它再次执行搜索，但是基于相关词而不是用户提供的原始关键词来查询匹配。

从应用程序的角度来看，当搜索结果太少时，可以使用查询扩展。再次执行搜索，但通过查询扩展为用户提供与他们正在查找的内容相关和相关的更多信息。

要使用查询扩展，请在 `AGAINST()` 函数中使用 `WITH QUERY EXPANSION` 搜索修饰符。以下说明使用 `WITH QUERY EXPANSION` 搜索修饰符查询的语法。

```
SELECT column1, column2
FROM table1
WHERE MATCH(column1, column2)
      AGAINST('keyword', WITH QUERY EXPANSION);
```

MySQL查询扩展示例

我们来看一下查询扩展的例子，看看它是如何工作的。

我们将使用 `products` 表的 `productName` 列来演示查询扩展功能。首先，[启用此列的全文搜索索引](#)。

```
ALTER TABLE products
ADD FULLTEXT(productName);
```

其次，搜索的产品名称包含 1992 的项，而不使用查询扩展。

```
SELECT productName  
FROM products  
WHERE MATCH(productName) AGAINST('1992');
```

执行上面查询语句，得到以下结果 -

如上所见，搜索结果其产品名称包含 1992 有 2 个产品。

第三，可以通过使用查询扩展来扩展搜索结果，如下所示：

```
SELECT productName  
FROM products  
WHERE MATCH(productName)  
      AGAINST('1992' WITH QUERY EXPANSION);
```

执行上面查询语句，得到以下结果 -

当我们使用查询扩展时，在搜索结果中得到了更多行。前两行是最相关的，其他行来自前两列的相关关键字，例如： Ferrari 。

请注意，通过返回不相关的结果，盲查询扩展会显着增加噪声。强烈建议您仅在搜索到的关键字较短时才使用查询扩展。

在本教程中，当用户提供的关键字很短时，我们向您介绍了MySQL查询扩展，以扩大搜索结果。

本教程将向您展示如何使用MySQL `ngram` 全文解析器来支持中文，日文，韩文等表意语言的全文搜索。

MySQL ngram全文解析器简介

MySQL内置的全文解析器使用空格确定单词的开始和结束。当涉及汉语，日语或韩语等表意语言语言时，这是一个限制，因为这些语言不使用分词符。

为了解决这个问题，MySQL提供了 `ngram` 全文解析器。自MySQL5.7.6版起，MySQL将 `ngram` 全文解析器作为内置的服务器插件，这意味着当MySQL数据库服务器启动时，MySQL会自动加载该插件。MySQL支持用于InnoDB和MyISAM存储引擎的 `ngram` 全文解析器。

根据定义，`ngram` 是来自文本序列的多个字符的连续序列。`ngram` 全文解析器的主要功能是将文本序列标记为 `n` 个字符的连续序列。

以下说明了 `ngram` 全文解析器如何标记不同值 `n` 的文本序列：

```
n = 1: 'm', 'y', 's', 'q', 'l'  
n = 2: 'my', 'ys', 'sq', 'ql'  
n = 3: 'mys', 'ysq', 'sql'  
n = 4: 'mysq', 'ysql'  
n = 5: 'mysql'
```

使用 `ngram` 解析器创建**FULLTEXT**索引

要创建使用 `ngram` 全文解析器的 `FULLTEXT` 索引，可以在[CREATE TABLE](#)，[ALTER TABLE](#)或 `CREATE INDEX` 语句中添加 `WITH PARSER ngram`。

例如，以下语句创建新的帖子表，并将标题和正文列添加到使用 `ngram` 全文解析器的 `FULLTEXT` 索引。

```
USE testdb;
CREATE TABLE posts (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255),
    body TEXT,
    FULLTEXT ( title , body ) WITH PARSER NGRAM
) ENGINE=INNODB CHARACTER SET UTF8;
```

以下INSERT语句向 posts 表中插入一个新行：

```
SET NAMES utf8;

INSERT INTO posts(title,body)
VALUES('MySQL全文搜索', 'MySQL提供了具有许多好的功能的内置全文搜索!'),
      ('MySQL教程', '学习MySQL快速，简单和有趣');
```

请注意，SET NAMES 语句设置客户端和服务器将用于发送和接收数据的字符集；在本示例中，它使用的是 utf8。

要查看 ngram 如何标记文本，请使用以下语句：

```
SET GLOBAL innodb_ft_aux_table="testdb/posts";

SELECT
*
FROM
information_schema.innodb_ft_index_cache
ORDER BY doc_id , position;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
*
FROM
information_schema.innodb_ft_index_cache
ORDER BY doc_id , position;
+-----+-----+-----+-----+-----+
-----+
```

WORD	FIRST_DOC_ID	LAST_DOC_ID	DOC_COUNT	DOC_ID	POSITION
my	2	3	2	2	2
0					
ys	2	3	2	2	2
1					
sq	2	3	2	2	2
2					
ql	2	3	2	2	2
3					
1金	2	2	1	2	2
4					
全文	2	2	1	2	2
5					
文搜	2	2	1	2	2
8					
搜索	2	2	1	2	2
11					
ql	2	3	2	2	2
18					
my	2	3	2	2	2
18					
ys	2	3	2	2	2
18					
sq	2	3	2	2	2
18					
1提	2	2	1	2	2
22					
提供	2	2	1	2	2
23					
供了	2	2	1	2	2
26					
了具	2	2	1	2	2
29					
具有	2	2	1	2	2
32					
有许	2	2	1	2	2
35					
许多	2	2	1	2	2
38					

多好	2	2	1	2
41				
好的	2	2	1	2
44				
的功	2	2	1	2
47				
功能	2	2	1	2
50				
能的	2	2	1	2
53				
的内	2	2	1	2
56				
内置	2	2	1	2
59				
搜索	2	2	1	2
60				
文搜	2	2	1	2
60				
全文	2	2	1	2
60				
置全	2	2	1	2
62				
my	2	3	2	3
0				
ys	2	3	2	3
1				
sq	2	3	2	3
2				
ql	2	3	2	3
3				
1教	3	3	1	3
4				
教程	3	3	1	3
5				
学习	3	3	1	3
12				
习m	3	3	1	3
15				
sq	2	3	2	3
18				
ql	2	3	2	3
18				

	2	3	2	3
my	2	3	2	3
18				
ys	2	3	2	3
18				
1快	3	3	1	3
22				
快速	3	3	1	3
23				
速，	3	3	1	3
26				
，简	3	3	1	3
29				
简单	3	3	1	3
32				
单和	3	3	1	3
35				
和有	3	3	1	3
38				
有趣	3	3	1	3
41				
-----+-----+-----+-----+-----+				
-----+-----+-----+-----+-----+				
50 rows in set				

此查询对于故障排除目的很有用。例如，如果一个单词不包括在搜索结果中，则该单词可能没有被编入索引，因为它是一个停止词或者可能是其它原因。

设置ngram令牌大小

在前面的示例可以看到，默认情况下， ngram 中的令牌大小(n)为 2 ，要更改令牌大小，请使用 ngram_token_size 配置选项，值的范围是： 1 到 10 。

请注意，较小的令牌大小可使较小的全文搜索索引更快地进行搜索。

因为 ngram_token_size 是只读变量，因此您只能使用两个选项设置其值：

第一种方式，在启动字符串中：

```
mysqld --ngram_token_size=1
```

第二种方式 - 在配置文件中：

```
[mysqld]
ngram_token_size=1
```

ngram解析器短语搜索

MySQL将短语搜索转换成 ngram 短语搜索。例如， abc 被转换为 ab bc ，它返回包含 ab bc 和 abc 的文档。

以下示例显示在 posts 表中搜索短语： 搜索 :

```
SELECT
    id, title, body
FROM
    posts
WHERE
    MATCH (title , body) AGAINST ('搜索' );
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
    id, title, body
FROM
    posts
WHERE
    MATCH (title , body) AGAINST ('搜索' );
+-----+
| id | title          | body           |
+-----+
| 1  | MySQL全文搜索 | MySQL提供了具有许多好的功能的内置全文搜索 |
+-----+
1 row in set
```

用ngram处理搜索结果

自然语言模式

在自然语言模式搜索中，搜索项被转换为 ngram 值的并集。假设令牌大小为 2 或者二进制，则搜索项 mysql 被转换为我的 my ys sq 和 ql 。

```
SELECT
*
FROM
posts
WHERE
    MATCH (title , body) AGAINST ('简单和有趣' IN natural language
MODE);
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
*
FROM
posts
WHERE
    MATCH (title , body) AGAINST ('简单和有趣' IN natural language
MODE);
+-----+-----+
| id | title      | body          |
+-----+-----+
| 2  | MySQL教程   | 学习MySQL快速，简单和有趣 |
+-----+-----+
1 row in set
```

布尔模式

在 BOOLEAN MODE 搜索中，搜索项被转换成 ngram 短语搜索。例如：

```

SELECT
*
FROM
posts
WHERE
    MATCH (title , body) AGAINST ('简单和有趣' IN BOOLEAN MODE);

```

执行上面查询语句，得到以下结果 -

```

mysql> SELECT
*
FROM
posts
WHERE
    MATCH (title , body) AGAINST ('简单和有趣' IN BOOLEAN MODE);
+-----+-----+
| id | title      | body          |
+-----+-----+
| 2  | MySQL教程   | 学习MySQL快速，简单和有趣 |
+-----+-----+
1 row in set

```

ngram通配符搜索

ngram `FULLTEXT` 索引仅包含 `ngram`，因此它不知道短语的开始。执行通配符搜索时，可能会返回意外的结果。

以下规则将应用于使用ngram `FULLTEXT` 搜索索引的通配符搜索：

如果通配符中的前缀短语短于 `ngram` 令牌大小，则查询返回所有包含以前缀项为起始的 `ngram` 令牌的文档。例如：

```

SELECT
    id, title, body
FROM
posts
WHERE
    MATCH (title , body) AGAINST ('my*' );

```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
    id, title, body
  FROM
    posts
 WHERE
    MATCH (title , body) AGAINST ('my*');
+-----+-----+
| id | title          | body
|-----+-----+
| 1  | MySQL全文搜索 | MySQL提供了具有许多好的功能的内置全文搜索 |
| 2  | MySQL教程       | 学习MySQL快速，简单和有趣
+-----+-----+
2 rows in set
```

如果通配符中的前缀短语长于 ngram 令牌大小，则MySQL将将前缀术语转换为 ngram 短语，并忽略通配符运算符。请参阅以下示例：

```
SELECT
  id, title, body
FROM
  posts
 WHERE
    MATCH (title , body) AGAINST ('mysqld*');
```

执行上面查询语句，得到以下结果 -

```

mysql> SELECT
    id, title, body
  FROM
    posts
 WHERE
    MATCH (title , body) AGAINST ('mysql*');
+-----+-----+
| id | title          | body
|-----+-----+
| 1  | MySQL全文搜索 | MySQL提供了具有许多好的功能的内置全文搜索 |
| 2  | MySQL教程       | 学习MySQL快速，简单和有趣
+-----+-----+
2 rows in set

```

在这个例子中，短语“mysql”被转换为 ngram 短语： my ys sq ql ld ，因此返回包含其中一个短语的所有文档。

处理停止词

ngram 解析器不包括在停止词列表中包含停止词的令牌。例如，假设 ngram_token_size 为 2 ，文档包含 abc 。 ngram 解析器将文档标记为 ab 和 bc 。如果 b 是一个停用词，则 ngram 将包含 ab 和 bc ，因为它们包含 b 。

请注意，如果语言不是英语，则必须定义自己的词条列表。此外，长度大于 ngram_token_size 的停止词将被忽略。

在本教程中，您已经学会了如何使用 MySQL ngram 全文解析器来处理表意语言的全文搜索。

第七章 函数

本节为您提供最常用的MySQL函数，包括聚合函数，字符串函数，日期时间函数，控制流函数等。

MySQL聚合函数

- [MySQL聚合函数](#) - 提供最常用的MySQL聚合函数的简要概述。
- [avg\(\)函数](#) - 计算一组值或表达式的平均值。
- [count\(\)函数](#) - 计算表中的行数。
- [instr\(\)函数](#) - 返回子字符串在字符串中第一次出现的位置。
- [sum\(\)函数](#) - 计算一组值或表达式的总和。
- [min\(\)函数](#) - 在一组值中找到最小值。
- [max\(\)函数](#) - 在一组值中找到最大值。
- [group_concat\(\)函数](#) - 将字符串从分组中连接成具有各种选项
(如 `DISTINCT` , `ORDER BY` 和 `SEPARATOR`)的字符串。
- [MySQL标准偏差函数](#) - 显示如何计算人口标准偏差和样本标准偏差。

MySQL字符串函数

- [concat\(\)函数](#) - 将两个或多个字符串组合成一个字符串。
- [length\(\)函数&char_length\(\)函数](#) - 以字节和字符获取字符串的长度。
- [left\(\)函数](#) - 获取指定长度的字符串的左边部分。
- [replace\(\)函数](#) - 搜索并替换字符串中的子字符串。
- [substring\(\)函数](#) - 从具有特定长度的位置开始提取一个子字符串。
- [trim\(\)函数](#) - 从字符串中删除不需要的字符。
- [find_in_set\(\)函数](#) - 在逗号分隔的字符串列表中找到一个字符串。
- [format\(\)函数](#) - 格式化具有特定区域设置的数字，舍入到小数位数。

MySQL控制流函数

- [case\(\)函数](#) - 如果满足 `WHEN` 分支中的条件，则返回 `THEN` 分支中的相应结果，否则返回 `ELSE` 分支中的结果。

- **if语句** - 根据给定的条件返回一个值。
- **ifnull()函数** - 如果第一个参数不为 `NULL` , 则返回第一个参数, 否则返回第二个参数。
- **nullif()函数** - 如果第一个参数等于第二个参数, 则返回 `NULL` , 否则返回第一个参数。

MySQL 日期和时间函数

- **curdate()函数** - 返回当前日期。
- **datediff()函数** - 计算两个 `DATE` 值之间的天数。
- **day()函数** - 获取指定日期月份的天(日)。
- **date_add()函数** - 将时间值添加到日期值。
- **date_sub()函数** - 从日期值中减去时间值。
- **date_format()函数** - 根据指定的日期格式格式化日期值。
- **dayname()函数** - 获取指定日期的工作日的名称。
- **dayofweek()函数** - 返回日期的工作日索引。
- **extract()函数** - 提取日期的一部分。
- **now()函数** - 返回当前日期和时间。
- **month()函数** - 返回一个表示指定日期的月份的整数。
- **str_to_date()函数** - 将字符串转换为基于指定格式的日期和时间值。
- **sysdate()函数** - 返回当前日期。
- **timediff()函数** - 计算两个 `TIME` 或 `DATETIME` 值之间的差值。
- **timestampdiff()函数** - 计算两个 `DATE` 或 `DATETIME` 值之间的差值。
- **week()函数** - 返回一个日期的星期数值。
- **weekday()函数** - 返回一个日期表示为工作日/星期几的索引。
- **year()函数** - 返回日期值的年份部分。

MySQL 比较函数

- **coalesce()函数** - 返回第一个非 `NULL` 参数, 这非常适合用于将值替换为 `NULL` 。
- **greatest()函数&least()函数** - 使用 `n` 个参数, 并分别返回 `n` 个参数的最大值和最小值。
- **isnull()函数** - 如果参数为 `NULL` , 则返回 `1` , 否则返回 `0` 。

其他MySQL函数

- [last_insert_id\(\)函数](#) - 获取最后插入的记录的最后生成的序列号。
- [cast\(\)函数](#) - 将任何类型的值转换为具有指定类型的值。

第八章 管理

在本节中，您将学习有关MySQL管理教程，包括MySQL服务器启动和关闭，MySQL服务器安全性，MySQL数据库维护和备份。

1. MySQL访问控制系统入门

- MySQL实现了一个复杂的访问控制和权限系统，允许您创建用于处理客户端操作的完整访问规则，并防止未经授权的客户端访问数据库系统。参考阅读：<http://www.yiibai.com/mysql/getting-started-with-mysql-access-control-system.html>

2. 如何使用MySQL的CREATE USER语句创建用户帐户

- 在本教程中，您将学习如何使用 CREATE USER 和 INSERT 语句在MySQL中创建用户。参考阅读：<http://www.yiibai.com/mysql/create-user.html>

3. 修改MySQL用户密码的3种最佳方法

- 在本教程中，您将通过使用 UPDATE ， SET PASSWORD 和 GRANT 语句，这三种方式学习如何更改或重置MySQL帐户的密码。参考阅读：<http://www.yiibai.com/mysql/grant.html>

4. 使用MySQL REVOKE撤销用户的权限

- 在本教程中，您将学习如何使用MySQL中的 REVOKE 语句来撤销MySQL帐户的权限。参考阅读：<http://www.yiibai.com/mysql/revoke.html>

5. MySQL角色使用指南

- 在本教程中，您将学习如何使用MySQL角色来简化权限管理。参考阅读：<http://www.yiibai.com/mysql/roles.html>

6. 如何使用MySQL DROP USER语句删除用户帐户

- 本教程将介绍如何使用 DROP USER 语句删除数据库服务器中的MySQL用户帐户。参考阅读：<http://www.yiibai.com/mysql/drop-user.html>

7. 维护MySQL数据库表

- MySQL提供了几个有用的语句，允许您维护数据库表以提高表访问的效率。这

些语句包括分析，优化，检查和修复表。参考阅读：<http://www.yiibai.com/mysql/database-table-maintenance-statements.html>

8.如何使用mysqldump工具备份数据库

- 在本教程中，您将学习如何使用 `mysqldump` 工具来备份MySQL数据库。参考阅读：<http://www.yiibai.com/mysql/how-to-backup-database-using-mysqldump.html>

9.如何列出MySQL中的所有数据库

- 在本教程中，您将学习如何使用MySQL的 `SHOW DATABASES` 命令列出MySQL数据库服务器中的数据库。参考阅读：<http://www.yiibai.com/mysql/show-databases.html>

10.如何列出MySQL数据库中的所有表

- 在本教程中，您将学习如何使用MySQL的 `SHOW TABLES` 命令查询指定数据库中的所有表。参考阅读：<http://www.yiibai.com/mysql/show-tables.html>

11.MySQL SHOW COLUMNS和DESCRIBE：列出表中的所有列

- 在本教程中，您将学习如何使用MySQL的 `DESCRIBE` 语句和 `SHOW COLUMNS` 命令来显示指定表的所有列。参考阅读：<http://www.yiibai.com/mysql/show-columns.html>

12.MySQL列出MySQL数据库服务器中的所有用户

- 您在寻找MySQL的 `SHOW USERS` 命令吗？不幸的是，MySQL没有 `SHOW USERS` 命令，但是您可以通过简单的查询来获取MySQL数据库中的所有用户。参考阅读：<http://www.yiibai.com/mysql/show-users.html>

在本教程中，我们将向您介绍MySQL访问控制系统和MySQL中各种与权限相关的表。

MySQL实现了复杂的访问控制和权限系统，允许您创建用于处理客户端操作的全面的访问规则，并有效防止未经授权的客户端访问数据库系统。

当客户端连接到服务器时，MySQL访问控制有两个阶段：

- 连接验证：连接到MySQL数据库服务器的客户端需要有一个有效的用户名和密码。此外，客户端连接的主机必须与MySQL授权表中的主机相匹配。
- 请求验证：当连接成功建立后，对于客户端发出的每个语句，MySQL会检查客户端是否具有足够的权限来执行该特定语句。MySQL能够检查数据库，表和字段级别的权限。

MySQL安装程序自动创建一个名为 `mysql` 的数据库。`mysql` 数据库包含五个主要的授权表。您可通过[GRANT](#)和[REVOKE](#)等语句间接操作这些表 -

- `user` 表：包含用户帐户和全局权限列。MySQL使用 `user` 表来接受或拒绝来自主机的连接。在 `user` 表中授予的权限对MySQL服务器上的所有数据库都有效。
- `db` 表：包含数据库级权限。MySQL使用数据库表来确定用户可以访问哪个数据库以及哪个主机。在 `db` 表中的数据库级授予的特权适用于数据库，所有对象属于该数据库，例如表，触发器，视图，存储过程等。
- `table_priv` 和 `columns_priv` 表：包含表级和列级权限。
在 `table_priv` 表中授予的权限适用于表及其列，而在 `columns_priv` 表中授予的权限仅适用于表的特定列。
- `procs_priv` 表：包含[存储函数](#)和[存储过程](#)的权限。

MySQL利用这些表来控制MySQL数据库服务器的权限。在实现自己的灵活访问控制系统之前，了解这些表非常重要。

在本教程中，您已经了解了MySQL访问控制系统的工作原理，并探索了MySQL中的授权表。

在本教程中，您将学习如何使用MySQL的 `CREATE USER` 语句在MySQL中创建一个用户帐户。

MySQL中的用户帐户简介

在MySQL中，不仅可以指定谁可以连接到数据库服务器，还可以指定用户连接的主机。因此，MySQL中的用户帐号由用户名，以及使用 `@` 字符分隔的主机名组成。

例如，如果 `admin` 用户从 `localhost` 主机连接到MySQL数据库服务器，则用户帐户是书写形式是：`admin@localhost`，其中 `@` 符号是一个固定的分隔符。

`admin` 用户只能从本地主机(`localhost`)连接到MySQL数据库服务器，而不是远程主机(如`:yiibai.com`)，这使得MySQL数据库服务器更加安全。

注意：如要限制 `admin` 用户只能从 `yiibai.com` 主机登录，那么可以书写为：`admin@yiibai.com`，如果想要允许从任意主机登录，那么可以书写为：`admin@%`。

此外，通过组合用户名和主机，可以设置多个具有相同名称的帐户，但可以从具有不同权限的不同主机进行连接。

MySQL将用户帐户授权存储在 `mysql` 数据库的 `user` 表中。

使用MySQL `CREATE USER`语句创建用户帐户

MySQL提供了 `CREATE USER` 语句，允许您创建一个新的用户帐户。`CREATE USER` 语句的语法如下：

```
CREATE USER user_account IDENTIFIED BY password;
```

用户帐号(`user_account`)是以 `username@hostname` 格式跟在 `CREATE USER` 子句之后。

密码(`password`)在 `IDENTIFIED BY` 子句中指定。`password` 必须是明文。在将用户帐户保存到用户表之前，MySQL将加密明文密码。

例如，要创建一个新的用户 `dbadmin`，这个用户只允许从 `localhost` 主机并使用密码为 `pwd123` 连接到MySQL数据库服务器，使用 `CREATE USER` 语句，如下所示：

```
CREATE USER dbadmin@localhost  
IDENTIFIED BY 'pwd123';
```

如果用户 `dbadmin` 还可以从IP为 `192.168.1.100` 的主机连接到MySQL数据库服务器，使用 `CREATE USER` 语句，如下所示：

```
CREATE USER dbadmin@192.168.1.100  
IDENTIFIED BY 'pwd123';
```

要查看用户帐户的权限，请使用 `SHOW GRANTS` 语句，如下所示：

```
SHOW GRANTS FOR dbadmin@localhost;
```

执行上面查询语句，得到以下结果 -

```
mysql> SHOW GRANTS FOR dbadmin@localhost;  
+-----+  
| Grants for dbadmin@localhost |  
+-----+  
| GRANT USAGE ON *.* TO 'dbadmin'@'localhost' |  
+-----+  
1 row in set
```

上面结果中的 `*.*` 显示 `dbadmin` 用户帐户只能登录到数据库服务器，没有其他权限。要授予用户权限，您可以使用[GRANT语句](#)，有关 `Grant` 语句，我们将在下一个教程中介绍。

请注意，点(.)之前的部分表示数据库，点(.)后面的部分表示表，例如 `database.table`，`testdb.offices` 等等。

要允许用户帐户从任何主机连接，请使用百分比(%)通配符，如以下示例所示：

```
CREATE USER superadmin@'%' IDENTIFIED BY 'mypassword';
```

百分比通配符 % 与 LIKE 运算符中使用的效果相同，例如，要允许 mysqladmin 用户帐户从 yiibai.com 主机的任何子域连接到数据库服务器，请使用百分比通配符 %，如下所示：

```
CREATE USER mysqladmin@'%.yiibai.com'  
IDENTIFIED by 'mypassword';
```

请注意，也可以在 CREATE USER 语句中使用下划线通配符 _。

如果您省略了用户帐户的主机名部分，MySQL 也会接受它，并允许用户从任何主机进行连接。例如，以下语句创建一个名为 remote_user 的新用户帐户，可以从任何主机连接到数据库服务器：

```
CREATE USER remote_user;
```

可以看到授予远程用户帐户(remote_user)的权限如下：

```
SHOW GRANTS FOR remote_user;
```

执行上面查询语句，得到以下结果 -

```
mysql> SHOW GRANTS FOR remote_user;  
+-----+  
| Grants for remote_user@% |  
+-----+  
| GRANT USAGE ON *.* TO 'remote_user'@'%' |  
+-----+  
1 row in set
```

要注意，引用是非常重要的，特别是当用户帐户包含特殊字符(如 _ 或 %)时。

如果您不小心引用 "username@hostname" 这样的用户帐户，MySQL 将创建一个 username@hostname 的用户，并允许用户从任何主机进行连接，这可能不是您预期的。

例如，以下语句创建可以从任何主机连接到MySQL数据库服务器的新用户 `api@localhost` (这是用户名，并非用户账号)。

```
-- 常用创建用户为：CREATE USER api@'localhost' 与下面语句不同  
CREATE USER 'api@localhost';
```

查看上面创建的用户的权限 -

```
mysql> SHOW GRANTS FOR 'api@localhost';  
+-----+  
| Grants for api@localhost@% |  
+-----+  
| GRANT USAGE ON *.* TO 'api@localhost'@'%' |  
+-----+  
1 row in set
```

如果创建一个已经存在的用户，MySQL会发出一个错误。例如，以下语句创建一个名为 `remote_user` 的用户帐户已经存在：

```
CREATE USER remote_user;
```

上面语句执行后，MySQL发出以下错误信息：

```
1396 - Operation CREATE USER failed for 'remote_user'@'%'
```

请注意，`CREATE USER` 语句只是创建一个没有任何权限的新用户帐户。如果要向用户授予使用权限，则使用 `GRANT` 语句。

在本教程中，您将学习如何使用MySQL中的 GRANT 语句向MySQL用户授予权限。

MySQL GRANT语句简介

创建新的用户帐户后，用户没有任何权限。如要向用户帐户授予权限，请使用 GRANT 语句。

下面说明 GRANT 语句的语法：

```
GRANT privilege,[privilege],... ON privilege_level  
TO user [IDENTIFIED BY password]  
[REQUIRE tsl_option]  
[WITH [GRANT_OPTION | resource_option]];
```

下面让我们来详细地看看 GRANT 语句 -

- 首先，在 GRANT 关键字之后指定一个或多个特权。如果要授予用户多个权限，则每个权限都将以逗号分隔(见下表中的特权列表)。
- 接下来，指定确定特权应用级别的 privilege_level 。MySQL 支持全局 (*.*)，数据库(database.*)，表(database.table)和列级别。如果您使用列权限级别，则必须在每个权限之后使用逗号分隔列的列表。
- 然后，放置要授予权限的用户。如果用户已经存在，则 GRANT 语句修改其特权。如不存在，则 GRANT 语句将创建一个新用户。可选的条件 IDENTIFIED BY 允许为用户设置新密码。
- 之后，可指定用户是否必须通过安全连接(如 SSL ， X059 等)连接到数据库服务器。
- 最后，可选的 WITH GRANT OPTION 子句允许此用户授予其他用户或从其他用户删除您拥有的权限。此外，可以使用 WITH 子句来分配MySQL数据库服务器的资源，例如，设置用户每小时可以使用多少个连接或语句。这在MySQL共享托管等共享环境中非常有用。

请注意，要使用 GRANT 语句，您必须具有 GRANT OPTION 权限和您授予其它用户的权限。如果启用了 read_only 系统变量，则需要具有 SUPER 权限才能执行 GRANT 语句授权。

我们来练习一些使用MySQL中的 GRANT 语句的例子来更好的理解。

MySQL GRANT示例

通常，我们首先使用 `CREATE USER` 语句创建新的用户帐户，然后再使用 `GRANT` 语句向用户授予权限。

例如，以下 `CREATE USER` 语句创建一个新的超级用户帐户。

```
CREATE USER super@localhost IDENTIFIED BY 'newpasswd';
```

要查看已分配给 `super@localhost` 用户帐户的权限，请使用 `SHOW GRANTS` 语句。

```
SHOW GRANTS FOR super@localhost;
```

上面代码执行结果如下 -

```
mysql> SHOW GRANTS FOR super@localhost;
+-----+
| Grants for super@localhost |
+-----+
| GRANT USAGE ON *.* TO 'super'@'localhost' |
+-----+
1 row in set
```

要向 `super@localhost` 用户帐户授予所有权限，请使用以下语句。

```
GRANT ALL ON *.* TO 'super'@'localhost' WITH GRANT OPTION;
```

`ON *.*` 子句表示MySQL中的所有数据库和所有对象。`WITH GRANT OPTION` 允许 `super@localhost` 向其他用户授予权限。

现在，如果再次执行 `SHOW GRANTS FOR super@localhost` 语句，您将看到 `super@localhost` 的权限已被更新。

```
SHOW GRANTS FOR super@localhost;
```

执行上面查询语句，得到以下结果 -

```
mysql> SHOW GRANTS FOR super@localhost;
+-----+
| Grants for super@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'super'@'localhost' WITH GRANT OPTION |
+-----+
| 1 row in set
```

要创建一个用户: `auditor`，并所有示例数据库(`yiibaidb`)中的所有权限，请使用以下语句：

```
CREATE USER auditor@localhost IDENTIFIED BY 'newpasswd';
GRANT ALL ON yiibaidb.* TO auditor@localhost;
```

您可以在单个 `GRANT` 语句中授予多个权限。例如，创建一个针对 `yiibaidb` 数据库执行`SELECT`，`INSERT`和`UPDATE`语句的权限的用户，如下语句：

```
CREATE USER rfc IDENTIFIED BY 'mypasswd';
GRANT SELECT, UPDATE, DELETE ON yiibaidb.* TO rfc;
```

现在，如果您使用 `rfc` 用户帐户登录到MySQL服务器并发出以下查询语句：

```
CREATE TABLE city(
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255)
);
```

MySQL将发出以下错误信息：

```
ERROR 1142 (42000): CREATE command denied to user 'rfc'@'localhost' for table 'city'
```

GRANT允许的特权

下表说明了可用于 GRANT 和 REVOKE 语句的所有可用权限：

权限	含义	全局	数据库	表	列	过程	代理
ALL [PRIVILEGES]	授予除了 GRANT OPTION 之外的指定访问级别的所有权限						
ALTER	允许用户使用 ALTER TABLE 语句	X	X	X			
ALTER ROUTINE	允许用户更改或删除存储程序	X	X			X	
CREATE	允许用户创建数据库和表	X	X	X			
CREATE ROUTINE	X	X					
CREATE TABLESPACE	允许用户创建，更改或删除表空间和日志文件组	X					
CREATE TEMPORARY TABLES	允许用户使用 CREATE TEMPORARY TABLE 创建临时表	X	X				
CREATE USER	允许用户使用 CREATE USER , DROP USER , RENAME USER 和 REVOKE ALL PRIVILEGES 语句。	X					
CREATE VIEW	允许用户创建或修改视图	X	X	X			
DELETE	允许用户使用 DELETE	X	X	X			
DROP	允许用户删除数据库，表和视图	X	X	X			
EVENT	能够使用事件计划的事件	X	X				
	允许用户执行存储过程/存						

	储函数					
FILE	允许用户读取数据库目录中的任何文件	X				
GRANT OPTION	允许用户有权授予或撤销其他帐户的权限	X	X	X		X X
INDEX	允许用户创建或删除索引	X	X	X		
INSERT	允许用户使用 INSERT 语句	X	X	X	X	
LOCK TABLES	允许用户在具有 SELECT 权限的表上使用 LOCK TABLES	X	X			
PROCESS	允许用户使用 SHOW PROCESSLIST 语句查看所有进程	X				
PROXY	启用用户代理					
REFERENCES	允许用户创建外键	X	X	X	X	
RELOAD	允许用户使用 FLUSH 操作	X				
REPLICATION CLIENT	允许用户查询主服务器或从服务器的位置	X				
REPLICATION SLAVE	允许用户使用复制从站从主机读取二进制日志事件	X				
SELECT	允许用户使用 SELECT 语句	X	X	X	X	
SHOW DATABASES	允许用户显示所有数据库	X				
SHOW VIEW	允许用户使用 SHOW CREATE VIEW 语句	X	X	X		
SHUTDOWN	允许用户使用 mysqladmin shutdown 命令	X				
SUPER	允许用户使用其他管理操作，如 CHANGE MASTER TO , KILL , PURGE BINARY LOGS , SET GLOBAL 和 mysqladmin 命令	X				
TRIGGER	允许用户使用 TRIGGER 操作	X	X	X		

TRIGGER	作	X	X	X			
UPDATE	允许用户使用 UPDATE 语句	X	X	X	X		
USAGE	相当于“无权限”						

在本教程中，您已经学会了如何使用MySQL的 GRANT 语句向用户授予权限。

在本教程中，您将学习如何使用MySQL中的 `REVOKE` 语句从MySQL帐户中撤销权限。

我们强烈建议您遵循以下教程，以更好地了解MySQL `REVOKE` 语句的工作原理：

- MySQL访问控制系统入门
- 如何创建一个MySQL用户
- 如何授予MySQL用户权限

MySQL `REVOKE`语句简介

要从用户帐户撤销权限，您可以使用MySQL `REVOKE` 语句。MySQL允许您从用户撤销一个或多个特权或所有权限。

以下说明从用户撤销指定权限的语法：

```
REVOKE  privilege_type [(column_list)]
      [, priv_type [(column_list)]]...
  ON [object_type] privilege_level
  FROM user [, user]...
```

我们来详细看看MySQL `REVOKE` 语句使用 -

- 首先，在 `REVOKE` 关键字之后指定要从用户撤销的权限列表，需要用逗号分隔权限。
- 其次，在 `ON` 子句中指定要撤销权限的权限级别。
- 第三，在 `FROM` 子句中指定要撤销的权限的用户帐户

请注意，要从用户帐户撤销权限，您必须具有 `GRANT OPTION` 权限和要撤销的权限。

要撤消用户的权限，请使用以下 `REVOKE` 语句：

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user]...
```

要执行 `REVOKE ALL` 语句，必须具有全局 `CREATE USER` 权限或 `mysql` 数据库的 `UPDATE` 权限。

要撤销代理用户，请使用 `REVOKE PROXY` 命令，如下所示：

```
REVOKE PROXY ON user FROM user [, user]...
```

代理用户是MySQL中有效的用户，可以模拟(假冒)另一个用户，因此代理用户拥有其模拟的用户的所有权限。

在撤消用户权限之前，最好通过使用 `SHOW GRANTS` 语句来检查用户拥有的权限的情况，如下所示：

```
SHOW GRANTS FOR user;
```

MySQL REVOKE示例

假设 `rfc` 用户对示例数据库(`yiibaidb`)具有`SELECT`，`UPDATE`和`DELETE`权限。现在，如果要从 `rfc` 用户撤消 `UPDATE` 和 `DELETE` 权限，可以按如下方式执行：

首先，使用 `SHOW GRANTS` 语句检查用户的权限：

```
-- 查看用户的当前权限
SHOW GRANTS FOR rfc;

-- 授予用户权限
GRANT SELECT, UPDATE, DELETE ON `yiibaidb`.* TO 'rfc'@'%';
```

执行上面查询语句，得到以下结果 -

Grants for rfc@%
GRANT USAGE ON *.* TO 'rfc'@'%'
GRANT SELECT, UPDATE, DELETE ON `yiibaidb`.* TO 'rfc'@'%'

请注意，确实按照授予用户权限的教程文章学习并操作后，您可以创建 `rfc` 帐户并向其授予 `SELECT`，`UPDATE` 和 `DELETE` 特权，如下所示：

```
-- 创建用户  
CREATE USER IF NOT EXISTS rfc IDENTIFIED BY 'newpasswd';  
  
-- 授予用户权限  
GRANT SELECT, UPDATE, DELETE ON yiibaidb.* TO rfc;
```

其次，从 `rfc` 用户撤销 `UPDATE` 和 `DELETE` 权限，参考以下语句：

```
REVOKE UPDATE, DELETE ON yiibaidb.* FROM rfc;
```

第三，可以使用 `SHOW GRANTS` 命令再次查看 `rfc` 用户的权限。

```
-- 查看用户权限  
SHOW GRANTS FOR 'rfc'@'%';  
-- 授予用户权限  
GRANT SELECT ON `yiibaidb`.* TO 'rfc'@'%';
```

如果要撤销 `rfc` 用户的所有权限，请执行以下命令：

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM rfc;
```

如果再次查看 `rfc` 用户的权限，您将看到 `rfc` 用户已经没有了权限。

```
-- 查看用户权限  
SHOW GRANTS FOR rfc;  
  
GRANT USAGE ON *.* TO 'rfc'@'%';
```

请注意，使用 `USAGE` 权限意味着MySQL中没有特权。

当 MySQL REVOKE 命令生效时

MySQL `REVOKE` 语句的生效时间取决于权限级别，如下所示：

- 客户端在后续会话中连接到MySQL时，对全局权限所做的更改才会生效。这些更改不适用于所有当前连接的用户。

- 数据库权限的更改将在下一个 USE 语句之后生效。
- 表和列权限的更改将在进行更改之后发出的所有查询时生效。

在本教程中，您学习了如何使用MySQL REVOKE 语句来撤销MySQL用户的权限。

在本教程中，您将学习如何使用MySQL角色来简化权限管理。

注意：本教程要求 MySQL 8+ 版本以上操作和执行，或自行参考：<http://dev.mysql.com/doc/refman/8.0/en/roles.html>

MySQL 角色简介

通常，MySQL 数据库拥有多个相同权限集合的用户。以前，向多个用户授予和撤销权限的唯一方法是单独更改每个用户的权限，假如用户数量比较多的时候，这是非常耗时的。

为了用户权限管理更容易，MySQL 提供了一个名为 `role` 的新对象，它是一个命名的特权集合。

如果要向多个用户授予相同的权限集，则应如下所示：

- 首先，创建新的角色。
- 第二，授予角色权限。
- 第三，授予用户角色。

如果要更改用户的权限，则需要仅更改授权角色的权限。这些更改角色的权限将对授予角色的所有用户生效。

MySQL 角色的例子

首先，创建一个名为 `crmdb` 的新数据库，用于存储客户关系管理数据。

```
CREATE DATABASE crmdb;
```

接下来，切换到 `crmdb` 数据库：

```
USE crmdb;
```

然后，在 `crmdb` 数据库中创建一个客户信息表：`customer`，其结构如下 -

```
USE crmdb;
CREATE TABLE `crmdb`.`customer`(
    id INT PRIMARY KEY AUTO_INCREMENT,
    first_name varchar(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    phone VARCHAR(32) NOT NULL,
    email VARCHAR(255)
);
```

之后，将一些数据插入到客户(customer)表中。

```
INSERT INTO customer(first_name, last_name, phone, email)
VALUES('Max', 'Su', '(+86)-0898-66887654', 'max.su@yiibai.com'),
      ('Lily', 'Bush', '(+86)-0898-66887985', 'lily.bush@yiibai.com')
);
```

最后，使用以下SELECT语句验证插入结果：

```
mysql> SELECT * FROM customer;
+-----+-----+-----+-----+-----+
| id | first_name | last_name | phone | email |
+-----+-----+-----+-----+-----+
| 1 | Max | Su | (+86)-0898-66887654 | max.su@yiibai.com |
| 2 | Lily | Bush | (+86)-0898-66887985 | lily.bush@yiibai.com |
+-----+-----+-----+-----+-----+
2 rows in set
```

创建角色

假设您开发了一个使用 `crmdb` 数据库的应用程序。要与 `crmdb` 数据库进行交互，您需要为需要完全访问数据库的开发人员创建帐户。此外，需要为仅需读取访问权限的用户创建帐户，以及为读取/写入访问权限的用户创建帐户。

要避免单独为每个用户帐户授予权限，您可以创建一组角色，并为每个用户帐户授予相应的角色。

要创建新角色，请使用 `CREATE ROLE` 语句，我们根据上面所述，一共要创建三个角色：

```
CREATE ROLE IF NOT EXISTS 'crm_dev', 'crm_read', 'crm_write';
```

角色名称类似于由用户和主机部分组成的用户帐户：`role_name@host_name`。

如果省略主机部分，则默认为“`%`”，表示任何主机。

授予角色权限

要授予角色权限，您可以使用 `GRANT` 语句。以下语句是向 `crm_dev` 角色授予 `crmdb` 数据库的所有权限：

```
GRANT ALL ON crmdb.* TO crm_dev;
```

以下语句授予 `crm_read` 角色 `SELECT` 权限：

```
GRANT SELECT ON crmdb.* TO crm_read;
```

以下语句赋予 `crm_write` 角色 `INSERT`，`UPDATE` 和 `DELETE` 权限：

```
GRANT INSERT, UPDATE, DELETE ON crm.* TO crm_write;
```

将角色分配给用户帐户

假设您需要一个用户帐户是开发人员，一个是具有只读访问权限的用户帐户和两个具有读/写访问权限的用户帐户。

要创建新用户，请使用 **CREATE USER** 语句，如下所示：

```
-- developer user  
CREATE USER `crm_dev1`@`localhost` IDENTIFIED BY 'passwd1990';  
-- read access user  
CREATE USER `crm_read1`@`localhost` IDENTIFIED BY 'passwd1990';  
-- read/write users  
CREATE USER `crm_write1`@`localhost` IDENTIFIED BY 'passwd1990';  
CREATE USER `crm_write2`@`localhost` IDENTIFIED BY 'passwd1990';
```

为了方便演示使用，所有用户密码都设置成一样的。

要为用户分配角色，请使用 **GRANT** 语句：

```
GRANT `crm_dev` TO `crm_dev1`@`localhost`;  
  
GRANT `crm_read` TO `crm_read1`@`localhost`;  
  
GRANT `crm_read`, `crm_write` TO `crm_write1`@`localhost`, `crm_write2`@`localhost`;
```

请注意，`crm_write1@localhost` 和 `crm_write2@localhost` 帐户的 **GRANT** 语句同时授予 `crm_read` 和 `crm_write` 角色。

要验证角色分配，请使用 **SHOW GRANTS** 语句，如下所示：

```
SHOW GRANTS FOR `crm_dev1`@`localhost`;
```

该语句返回以下结果集：

```
+-----+  
| Grants for `crm_dev1`@`localhost` |  
+-----+  
| GRANT USAGE ON *.* TO `crm_dev1`@`localhost` |  
| GRANT `crm_dev`@`%` TO `crm_dev1`@`localhost` |  
+-----+  
2 rows in set (0.02 sec)
```

正如你所看到的，它只返回授予角色。要显示角色所代表的权限，请使用 `USING` 子句和授权角色的名称，如下所示：

```
SHOW GRANTS FOR crm_write1@localhost USING crm_write;
```

该语句返回以下输出：

```
+-----+
-----+
| Grants for crm_write1@localhost
|
+-----+
-----+
| GRANT USAGE ON *.* TO `crm_write1`@`localhost`
|
| GRANT INSERT, UPDATE, DELETE ON `crm`.* TO `crm_write1`@`localhost`
|
| GRANT `crm_read`@`%`, `crm_write`@`%` TO `crm_write1`@`localhost`
|
+-----+
-----+
```

设置默认角色

现在，如果您使用 `crm_read1` 用户帐户连接到MySQL，并尝试访问 `yiibaidb` 数据库：

```
mysql -u crm_read1 -p
Enter password: *****
mysql>USE crmdb;
```

上面语句发出以下错误信息：

```
ERROR 1044 (42000): Access denied for user 'crm_read1'@'localhost'
  to database 'crmdb'
```

这是因为在向用户帐户授予角色时，当用户帐户连接到数据库服务器时，它不会自动使角色变为活动状态。

如果调用 `CURRENT_ROLE()` 函数：

```
SELECT current_role();  
  
+-----+  
| current_role() |  
+-----+  
| NONE          |  
+-----+  
1 row in set (0.00 sec)
```

它返回 `NONE`，意味着没有启用角色。

要在每次用户帐户连接到数据库服务器时指定哪些角色应该处于活动状态，请使用 `SET DEFAULT ROLE` 语句。

以下语句为 `crm_read1@localhost` 帐户的所有分配角色设置默认值。

```
SET DEFAULT ROLE ALL TO crm_read1@localhost;
```

现在，如果当使用 `crm_read1` 用户帐户连接到MySQL数据库服务器并调用 `CURRENT_ROLE()` 函数：

```
> mysql -u crm_read1 -p  
Enter password: *****  
> SELECT CURRENT_ROLE();
```

您将看到 `crm_read1` 用户帐户的默认角色，如下所示 -

可以通过将当前数据库切换到 `crmdb` 数据库，执行 `SELECT` 语句和 `DELETE` 语句来测试 `crm_read` 帐户的权限，如下所示：

```

mysql> use crmdb;
Database changed
mysql> SELECT COUNT(*) FROM customer;
+-----+
| COUNT(*) |
+-----+
|      2   |
+-----+
1 row in set (0.00 sec)

mysql> DELETE FROM customer;
ERROR 1142 (42000): DELETE command denied to user 'crm_read1'@'localhost' for table 'customer'

```

如上面结果所示，它的确按预期那样工作。当我们发出 `DELETE` 语句时，就收到一个错误，因为 `crm_read1` 用户帐户只能读取访问权限。

设置活动角色

用户帐户可以通过指定哪个授权角色处于活动状态来修改当前用户在当前会话中的有效权限。

以下语句将活动角色设置为 `NONE`，表示没有活动角色。

```
SET ROLE NONE;
```

要将活动角色设置为所有授予的角色，请使用：

```
SET ROLE ALL;
```

要将活动角色设置为由 `SET DEFAULT ROLE` 语句设置的默认角色，请使用：

```
SET ROLE DEFAULT;
```

要设置活动的命名角色，请使用：

```
SET ROLE granted_role_1, granted_role_2, ...;
```

撤销角色的权限

要从特定角色撤销权限，请使用 `REVOKE` 语句。`REVOKE` 语句不仅起到角色的作用，而且也赋予任何授予角色的帐户。

例如，要临时使所有读/写用户只读，您可以更改 `crm_write` 角色，如下所示：

```
REVOKE INSERT, UPDATE, DELETE ON crmdb.* FROM crm_write;
```

要恢复权限，需要重新授予它们权限，如下所示：

```
GRANT INSERT, UPDATE, DELETE ON crmdb.* FOR crm_write;
```

删除角色

要删除一个或多个角色，请使用 `DROP ROLE` 语句，如下所示：

```
DROP ROLE role_name, role_name, ...;
```

像 `REVOKE` 语句一样，`DROP ROLE` 语句从其授予的每个用户帐户中撤销角色。

例如，要删除 `crm_read`，`crm_write` 角色，请使用以下语句：

```
DROP ROLE crm_read, crm_write;
```

将权限从用户帐户复制到另一个用户

MySQL 将用户帐户视为角色，因此，可以将用户帐户授予另一个用户帐户，例如向该用户帐户授予角色。这允许将用户的权限复制到另一个用户。

假设您需要 `crmdb` 数据库的另一个开发人员帐户：

首先，创建新的用户帐户：

```
CREATE USER crm_dev2@localhost IDENTIFIED BY 'passwd1990';
```

其次，将 `crm_dev1` 用户帐户的权限复制到 `crm_dev2` 用户帐户，如下所示：

```
GRANT crm_dev1@localhost TO crm_dev2@localhost;
```

在本教程中，您已经学会了如何使用MySQL角色来管理用户帐户的权限。

在本教程中，您将学习如何使用MySQL `DROP USER` 语句来删除用户帐户。

MySQL DROP USER语句介绍

要删除一个或多个用户帐户，请按如下所示使用 `DROP USER` 语句：

```
DROP USER user, [user], ...;
```

要删除用户，可以在 `DROP USER` 子句之后的 '`user_name'@'host_name'`' 格式中指定帐户名称。如果要一次删除多个用户，请使用以逗号分隔的用户列表。

如果删除不存在的用户帐户，MySQL将发出错误。从MySQL 5.7.8开始，可以使用 `IF EXISTS` 子句来指示该语句在删除不存在的用户帐户时发出警告，而不是发出错误。

```
DROP USER [IF EXISTS] user, [user], ...;
```

除了删除用户帐户之外，`DROP USER` 语句还会从所有授权表中删除所有权限。

删除用户帐户示例

要查看MySQL服务器中的数据库：`mysql` 的所有用户信息，请使用以下`SELECT`语句：

```
USE mysql;  
SELECT user, host FROM mysql.user;
```

下面是我在编写本教程时所使用MySQL数据库中的用户帐号列表：

```
mysql> SELECT user, host FROM mysql.user;
+-----+-----+
| user      | host   |
+-----+-----+
| api@localhost | %    |
| remote_user  | %    |
| rfc          | %    |
| superadmin   | %    |
| auditor     | localhost |
| dbadmin      | localhost |
| mysql.sys    | localhost |
| root         | localhost |
| super        | localhost |
+-----+-----+
9 rows in set
```

假设您要删除用户帐户：`api@localhost`，请使用以下语句：

```
DROP USER 'api@localhost';
```

再次从 `mysql.user` 表中查询数据，您将看到用户 `api@localhost` 已被删除。

数据库中的 `mysql.user` 表的用户帐号列表如下所示：

```
mysql> SELECT user, host FROM mysql.user;
+-----+-----+
| user      | host   |
+-----+-----+
| remote_user | %    |
| rfc          | %    |
| superadmin   | %    |
| auditor     | localhost |
| dbadmin      | localhost |
| mysql.sys    | localhost |
| root         | localhost |
| super        | localhost |
+-----+-----+
8 rows in set
```

要删除单个 `DROP USER` 语句中两个用户：`remote_user` 和 `auditor` 的帐户，请使用以下语句：

```
DROP USER remote_user, auditor;
```

下面可验证删除操作的结果，再次查询 `mysql.user` 表中的用户信息 -

```
SELECT user, host FROM mysql.user;
```

假设用户帐户已登录，并且正在运行会话。如果您删除用户帐户，则不会停止打开的会话。活动会话将继续，直到用户退出。通常，在这种情况下，您应该在执行 `DROP USER` 语句之前立即关闭用户的会话。

首先，需要使用 `SHOW PROCESSLIST` 语句识别用户的进程标识。

如您所见，`dbadmin@localhost` 用户帐户的进程号为：`40`。

第二，通过杀死这个进程 -

```
KILL 40;
```

用户帐户 `dbadmin@localhost` 收到一条错误消息：

```
ERROR 2013 (HY000): Lost connection to MySQL server during query
```

第三，执行 `DROP USER` 语句删除用户帐户 `dbadmin@localhost`：

```
DROP USER dbadmin@localhost;
```

请注意，如果不终止活动会话，则删除的用户(如果连接到数据库服务器)仍然可以执行所有操作，直到会话结束。

在本教程中，您已经学会了如何使用MySQL `DROP USER` 语句来删除一个或多个用户帐户。

在本教程中，我们将向您介绍一些非常有用的语句，使您可以在MySQL中维护数据库表。

MySQL提供了几个有用的语句，可以有效地维护数据库表。这些语句使您能够分析，优化，检查和修复数据库表。

分析表语句

MySQL查询优化器是MySQL服务器的重要组成部分，为查询创建了一个最佳的查询执行计划。对于特定查询，查询优化器使用存储的密钥分发和其他因素来决定在执行连接时应将表进行连接的顺序，以及哪个索引应用于特定表。

然而，密钥分发可能有时不准确，例如，在表中进行了大量数据更改(包括插入，删除或更新)之后。如果密钥分发不准确，则查询优化器可能会选择可能导致严重性能问题的错误查询执行计划。

要解决此问题，您可以为该表运行 `ANALYZE TABLE` 语句，例如，以下语句分析示例数据库(yiibaidb)中的 `payments` 表。如下所示 -

```
mysql> ANALYZE TABLE payments;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| yiibaidb.payments | analyze | status    | OK      |
+-----+-----+-----+-----+
1 row in set
```

如果运行 `ANALYZE TABLE` 语句后表没有变化，MySQL将不会再次分析表。如果再次运行上述语句：

```
mysql> ANALYZE TABLE payments;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| yiibaidb.payments | analyze | status    | OK      |
+-----+-----+-----+-----+
1 row in set
```

当前表已经是最新的状态了。

优化表语句

在使用数据库时，可以进行许多更改，如插入，删除或更新表中的数据，这可能会导致表的物理存储碎片化。因此，数据库服务器的性能下降。

MySQL提供了一个语句，允许您优化表以避免此碎片整理问题。以下说明如何优化表：

```
OPTIMIZE TABLE table_name;
```

建议您对经常更新的表执行此语句。例如，如果要优化 `orders` 表进行碎片整理，可以执行以下语句：

```
mysql> OPTIMIZE TABLE orders;
+-----+-----+-----+
| Table | Op      | Msg_type | Msg_text
+-----+-----+-----+
| yiibaidb.orders | optimize | note      | Table does not support
| yiibaidb.orders | optimize | status    | OK
+-----+-----+-----+
2 rows in set
```

检查表语句

数据库服务器可能会发生错误，例如，服务器意外关闭，将数据写入硬盘时出错，等等。这些情况可能导致数据库运行不正确，并且在最坏的情况下可能会崩溃。

MySQL允许您使用 `CHECK TABLE` 语句检查数据库表的完整性。以下说明 `CHECK TABLE` 语句的语法：

```
CHECK TABLE table_name;
```

CHECK TABLE 语句检查表及其索引。例如，可以使用 CHECK TABLE 语句来检查订单表，如下所示：

```
CHECK TABLE orders;
```

执行上面语句，结果如下 -

```
mysql> CHECK TABLE orders;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| yiibaidb.orders | check | status   | OK      |
+-----+-----+-----+-----+
1 row in set
```

CHECK TABLE 语句仅检测数据库表中的问题，但不会修复它们。要修复表，请使用 REPAIR TABLE 语句。

修复表语句

REPAIR TABLE 语句允许您修复数据库表中发生的一些错误。MySQL 不保证 REPAIR TABLE 语句可以修复表所有可能的错误。

以下是 REPAIR TABLE 语句的语法：

```
REPAIR TABLE table_name;
```

假设 orders 表中有一些错误，需要修复它们，可以使用 REPAIR TABLE 语句，如下查询所示：

```
REPAIR TABLE employees;
```

MySQL 返回表中所做的内容，并显示表是否已修复。

注： `REPAIR TABLE` 方法仅适用于 MyISAM，ARCHIVE 和 CSV 表。

在本教程中，您已经学到了一些非常方便的语句来维护MySQL中的数据库表。

在本教程中，您将学习如何使用mysqldump工具备份MySQL数据库。

MySQL GUI工具(如phpMyAdmin，SQLyog等)通常为备份MySQL数据库提供了方便的功能。但是，如果您的数据库很大，则备份过程可能非常慢，因为备份文件需要通过网络传输到客户端PC。因此，备份过程增加了MySQL数据库服务器的锁定和可用时间。

MySQL提供了非常有用的工具，用于在服务器上本地备份或转储MySQL数据库。备份文件存储在服务器中的文件系统中，因此您只需在需要时下载即可。

备份MySQL数据库的工具是 `mysqldump`。它位于MySQL安装文件夹的根 `/bin` 文件夹中。如本教程安装的位置为：`D:\software\mysql-5.7.18-winx64\bin\mysqldump.exe`

`mysqldump` 是由MySQL提供的程序，可用于转储数据库以备数据库或将数据库传输到另一个数据库服务器。

转储文件包含一组用于创建数据库对象的SQL语句。此外，`mysqldump` 可用于生成CSV，分隔符或XML文件。在本教程中，我们将仅关注如何使用 `mysqldump` 工具备份MySQL数据库。

在本教程中，我们将仅关注如何使用 `mysqldump` 工具备份MySQL数据库。

如何备份MySQL数据库

要备份MySQL数据库，数据库首先必须存在于数据库服务器中，并且您也可以访问该服务器。如果没有远程桌面，可以使用SSH或Telnet登录到远程服务器。备份MySQL数据库的命令如下：

```
mysqldump -u [username] -p[password] [database_name] > [dump_file.sql]
```



上述命令的参数如下：

- `[username]`：有效的MySQL用户名。
- `[password]`：用户的有效密码。请注意，`-p` 和密码之间没有空格。
- `[database_name]`：要备份的数据库名称
- `[dump_file.sql]`：要生成的转储文件。

通过执行上述命令，所有数据库结构和数据将导出到一个 [dump_file.sql] 转储文件中。例如，要备份示例数据库 yiibaidb，可使用以下命令：

```
mysqldump -u root -p123456 yiibaidb > D:\worksp\bakup\yiibaidb001.sql
```

执行上面语句，如下所示 -

```
C:\Users\Administrator> mysqldump -u root -p123456 yiibaidb > D:\worksp\bakup\yiibaidb001.sql
mysqldump: [Warning] Using a password on the command line interface can be insecure.
```

```
C:\Users\Administrator>
```

执行后，将创建一个文件： D:\worksp\bakup\yiibaidb001.sql

如何仅备份MySQL数据库结构

如果只想备份数据库结构而不需要备份数据，那么只需要添加一个选项 `-no-data` 来告诉 `mysqldump` 只需要导出数据库结构，如下：

```
mysqldump -u [username] -p[password] -no_data [database_name] > [dump_file.sql]
```

例如，仅使用结构来备份示例数据库，可以使用以下命令：

```
C:\Users\Administrator> mysqldump -u root -p123456 -no_data yiibaidb > D:\worksp\bakup\backup002.sql
```

如何仅备份MySQL数据库数据

有一种情况，您希望在分段和开发系统中刷新数据，因此这些系统中的数据与生产系统相同。

在这种情况下，只需要从生产系统导出数据，并将其导入到临时或开发系统中。要实现只备份数据，您可以使用 `mysqldump` 的选项 `-no-create-info`，如下所示：

```
mysqldump -u [username] -p[password] -no-create-info [database_name] > [dump_file.sql]
```

例如，要仅使用数据来备份示例数据库(`yiibaidb`)，请使用以下命令：

```
mysqldump -u root -p123456 -no-create info yiibaidb > D:\worksp\backup\backup003.sql
```

如何将多个MySQL数据库备份到一个文件中

如果要通过 `[database_name]` 中的命令来备份多个数据库，只需单独的数据库名称即可。如果要备份数据库服务器中的所有数据库，请使用选项 `-all-database`。

```
mysqldump -u [username] -p[password] [dbname1, dbname2, ...] > [all_dbs_dump_file.sql]
```

```
mysqldump -u [username] -p[password] -all-database > [all_dbs_dump_file.sql]
```

在本教程中，您已经学习了如何使用 `mysqldump` 工具来备份具有多种选项的 MySQL 数据库。

在本教程中，您将学习如何使用MySQL `SHOW DATABASES` 命令列出MySQL数据库服务器中的所有数据库。

使用MySQL SHOW DATABASES

要列出MySQL服务器主机上的所有数据库，请使用 `SHOW DATABASES` 命令，如下所示：

```
SHOW DATABASES;
```

例如，要列出本地MySQL数据库服务器中的所有数据库，请首先登录到数据库服务器，如下所示：

```
C:\Users\Administrator mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.9 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

然后使用 `SHOW DATABASES` 命令：

列出数据库

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| crmdb |
| mysql |
| newdb |
| performance_schema |
| testdb |
| yiibaidb |
| yiibaidb_backup |
+-----+
8 rows in set
```

`SHOW SCHEMAS` 命令是 `SHOW DATABASES` 的同义词，因此以下命令将返回与上述相同的结果：

```
mysql> SHOW SCHEMAS;
+-----+
| Database |
+-----+
| information_schema |
| crmdb |
| mysql |
| newdb |
| performance_schema |
| testdb |
| yiibaidb |
| yiibaidb_backup |
+-----+
8 rows in set
```

如果要查询与特定模式匹配的数据库，请使用 `LIKE` 子句，如下所示：

```
SHOW DATABASES LIKE pattern;
```

例如，以下语句返回以字符串“ schema ”结尾的数据库；

```
mysql> SHOW DATABASES LIKE '%schema';
+-----+
| Database (%schema) |
+-----+
| information_schema |
| performance_schema |
+-----+
2 rows in set
```

重要的是要注意，如果MySQL数据库服务器以 `-skip-show-database` 启动，则除非具有 `SHOW DATABASES` 权限，否则不能使用 `SHOW DATABASES` 语句。

从`information_schema`查询数据库数据

如果 `LIKE` 子句中的条件不足，可以直接从 `information_schema` 数据库中的 `schemata` 表查询数据库信息。

例如，以下查询返回与 `SHOW DATABASES` 命令相同的结果。

```
SELECT schema_name
FROM information_schema.schemata;
```

以下`SELECT`语句返回名称以' schema '或' db '结尾的数据库。

```
SELECT schema_name
FROM information_schema.schemata
WHERE schema_name LIKE '%schema' OR
      schema_name LIKE '%db';
```

它返回以下结果集：

```
mysql> SELECT schema_name
    FROM information_schema.schemata
   WHERE schema_name LIKE '%schema' OR
         schema_name LIKE '%db';
+-----+
| schema_name |
+-----+
| information_schema |
| crmdb |
| newdb |
| performance_schema |
| testdb |
| yiibaidb |
+-----+
6 rows in set
```

在本教程中，您已经学习了如何使用 `SHOW DATABASES` 命令显示MySQL服务器中的所有数据库，或者从 `information_schema` 数据库中的 `schemata` 表进行查询。

在本教程中，您将学习如何使用MySQL `SHOW TABLES` 命令查询特定数据库中的表。

要在MySQL数据库中列出所有表，请按照下列步骤操作：

1. 使用MySQL客户端(如 `mysql`)登录到MySQL数据库服务器
2. 使用 `USE` 语句切换到特定的数据库。
3. 使用 `SHOW TABLES` 命令。

下面说明了MySQL `SHOW TABLES` 命令的语法：

```
SHOW TABLES;
```

MySQL SHOW TABLES示例

以下示例说明如何列出 `yiibaidb` 数据库中的所有表。

步骤1 - 连接到MySQL数据库服务器：

```
C:\Users\Administrator mysql -u root -p
```

步骤2 - 切换到 `yiibaidb` 数据库：

```
mysql> USE yiibaidb;
Database changed
```

步骤3 - 显示 `yiibaidb` 数据库中的所有表：

```
mysql> show tables;
+-----+
| Tables_in_yiibaidb |
+-----+
| aboveavgproducts
| article_tags
| bigsalesorder
| contacts
| customerorders
| customers
| departments
| employees
| employees_audit
| officeinfo
| offices
| offices_bk
| offices_usa
| orderdetails
| orders
| organization
| payments
| price_logs
| productlines
| products
| saleperorder
| user_change_logs
| v_contacts
| vps
+-----+
24 rows in set
```

SHOW TABLES 命令可显示表是基表还是视图。要在结果中包含表类型，请使用 SHOW TABLES 语句，如下所示 -

```
SHOW FULL TABLES;
```

执行上面语句，如下所示 -

```
mysql> SHOW FULL TABLES;
+-----+-----+
| Tables_in_yiibaidb | Table_type |
+-----+-----+
| aboveavgproducts | VIEW
| article_tags | BASE TABLE
| bigsalesorder | VIEW
| contacts | BASE TABLE
| customerorders | VIEW
| customers | BASE TABLE
| departments | BASE TABLE
| employees | BASE TABLE
| employees_audit | BASE TABLE
| officeinfo | VIEW
| offices | BASE TABLE
| offices_bk | BASE TABLE
| offices_usa | BASE TABLE
| orderdetails | BASE TABLE
| orders | BASE TABLE
| organization | VIEW
| payments | BASE TABLE
| price_logs | BASE TABLE
| productlines | BASE TABLE
| products | BASE TABLE
| saleperorder | VIEW
| user_change_logs | BASE TABLE
| v_contacts | VIEW
| vps | VIEW
+-----+-----+
24 rows in set
```

我们在 `yiibaidb` 数据库中创建一个名为 `view_contacts` 的视图，其中包括来自 `employees` 和 `customers` 表的名字，姓氏和电话。

```
CREATE VIEW view_contacts
AS
SELECT lastName, firstName, extension AS phone
FROM employees
UNION
SELECT contactFirstName, contactLastName, phone
FROM customers;
```

现在，执行查询 SHOW FULL TABLES 命令：

```
mysql> SHOW FULL TABLES;
+-----+-----+
| Tables_in_yiibaidb | Table_type |
+-----+-----+
| aboveavgproducts | VIEW
| article_tags | BASE TABLE
| bigsalesorder | VIEW
| contacts | BASE TABLE
| customerorders | VIEW
| customers | BASE TABLE
| departments | BASE TABLE
| employees | BASE TABLE
| employees_audit | BASE TABLE
| officeinfo | VIEW
| offices | BASE TABLE
| offices_bk | BASE TABLE
| offices_usa | BASE TABLE
| orderdetails | BASE TABLE
| orders | BASE TABLE
| organization | VIEW
| payments | BASE TABLE
| price_logs | BASE TABLE
| productlines | BASE TABLE
| products | BASE TABLE
| saleperorder | VIEW
| user_change_logs | BASE TABLE
| v_contacts | VIEW
| view_contacts | VIEW
| vps | VIEW
+-----+-----+
25 rows in set
```

您可以看到，`v_contacts`, `view_contacts`, `vps` 等是视图(`VIEW`)，而其它表则都是基表(`BASE TABLE`)。

对于具有很多表的数据库，一次显示所有表可能不免直观。

幸运的是，`SHOW TABLES` 命令提供了一个选项，允许使用`LIKE`运算符或`WHERE`子句中的表达式对返回的表进行过滤，如下所示：

```
SHOW TABLES LIKE pattern;  
SHOW TABLES WHERE expression;
```

例如，要显示 yiibaidb 数据库中以字母 p 开头的所有表，请使用以下语句：

```
mysql> SHOW TABLES LIKE 'p%';  
+-----+  
| Tables_in_yiibaidb (p%) |  
+-----+  
| payments  
| price_logs  
| productlines  
| products  
+-----+  
4 rows in set
```

或者显示以' es '字符串结尾的表，可使用以下语句：

```
mysql> SHOW TABLES LIKE '%es';  
+-----+  
| Tables_in_yiibaidb (%es) |  
+-----+  
| employees  
| offices  
| productlines  
+-----+  
3 rows in set
```

以下语句说明了如何在 SHOW TABLES 语句中使用 WHERE 子句列出 yiibai 数据库中的所有视图 -

列出表

```
mysql> SHOW FULL TABLES WHERE table_type = 'VIEW';
+-----+-----+
| Tables_in_yiibaidb | Table_type |
+-----+-----+
| aboveavgproducts | VIEW
| bigsalesorder | VIEW
| customerorders | VIEW
| officeinfo | VIEW
| organization | VIEW
| saleperorder | VIEW
| v_contacts | VIEW
| view_contacts | VIEW
| vps | VIEW
+-----+
9 rows in set
```

有时，希望看到非当前使用的数据库中的表。可以使用 `SHOW TABLES` 语句的 `FROM` 子句来指定要显示表的数据库。

以下示例演示如何显示以' time '开头的表;

```
mysql> SHOW TABLES FROM mysql LIKE 'time%';
+-----+
| Tables_in_mysql (time%) |
+-----+
| time_zone
| time_zone_leap_second
| time_zone_name
| time_zone_transition
| time_zone_transition_type
+-----+
5 rows in set
```

以下语句相当于上面的语句，但它使用 `IN` 而不是 `FROM` 子句，如下所示 -

列出表

```
mysql> SHOW TABLES IN mysql LIKE 'time%';
+-----+
| Tables_in_mysql (time%) |
+-----+
| time_zone
| time_zone_leap_second
| time_zone_name
| time_zone_transition
| time_zone_transition_type |
+-----+
5 rows in set
```

请注意，如果您没有基表或视图的权限，则它不会显示在 `SHOW TABLES` 命令的结果集中。

在本教程中，您已经学习了如何使用MySQL `SHOW TABLES` 语句列出指定数据库中的所有表。

在本教程中，您将学习如何使用 `DESCRIBE` 语句和 MySQL `SHOW COLUMNS` 命令来显示表的列。

使用 `DESCRIBE` 语句

要显示表的所有列，请使用以下步骤：

1. 登录到 MySQL 数据库服务器
2. 切换到特定数据库(使用 `USE` 语句)
3. 使用 `DESCRIBE` 语句

以下示例演示如何在 `yiibaidb` 数据库中显示 `orders` 表的所有列。

步骤1 - 登录到MySQL数据库。

```
C:\Users\Administrator mysql -u root -p
```

步骤2 - 发出 `USE` 命令将数据库切换到 `yiibaidb` 数据库：

```
mysql> USE yiibaidb;
Database changed
```

步骤3 - 使用 `DESCRIBE` 语句，得到以下结果 -

```
mysql> DESCRIBE orders;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| orderNumber | int(11) | NO | PRI | NULL | 
| orderDate | date | NO | | NULL | 
| requiredDate | date | NO | | NULL | 
| shippedDate | date | YES | | NULL | 
| status | varchar(15) | NO | | NULL | 
| comments | text | YES | | NULL | 
| customerNumber | int(11) | NO | MUL | NULL | 
+-----+-----+-----+-----+-----+
7 rows in set
```

列出表的列

实际上，一般使用 `DESC` 语句，它是 `DESCRIBE` 语句的缩写。例如，以下语句等同于上面的 `DESCRIBE`，如下所示 -

```
mysql> DESC orders;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| orderNumber | int(11) | NO | PRI | NULL |
| orderDate | date | NO | | NULL |
| requiredDate | date | NO | | NULL |
| shippedDate | date | YES | | NULL |
| status | varchar(15) | NO | | NULL |
| comments | text | YES | | NULL |
| customerNumber | int(11) | NO | MUL | NULL |
+-----+-----+-----+-----+-----+
7 rows in set
```

MySQL SHOW COLUMNS命令

获取表中列的更灵活的方法是使用MySQL `SHOW COLUMNS` 命令。

```
SHOW COLUMNS FROM table_name;
```

要显示表的列，请在 `SHOW COLUMNS` 语句的 `FROM` 子句中指定表名。要显示指定数据库中不是当前数据库中的表的列定义信息，请使用以下形式：

```
SHOW COLUMNS FROM database_name.table_name;
-- 例如
SHOW COLUMNS FROM mysql.user;
```

或者 -

```
SHOW COLUMNS FROM table_name IN database_name;
-- 例如
SHOW COLUMNS FROM user IN mysql;
```

例如，要获取 `orders` 表的列，请使用 `SHOW COLUMNS` 语句，如下所示：

```
SHOW COLUMNS FROM orders;
```

您可以看到这个 `SHOW COLUMNS` 命令的结果与 `DESC` 语句的结果相同。

要获取有关列的更多信息，请将 `FULL` 关键字添加到 `SHOW COLUMNS` 命令中，如下所示：

```
SHOW FULL COLUMNS FROM table_name;
```

例如，以下语句列出了 `yiibaidb` 数据库中的 `payments` 表的所有列。

```
mysql> SHOW FULL COLUMNS FROM payments \G;
***** 1. row ****
Field: customerNumber
Type: int(11)
Collation: NULL
Null: NO
Key: PRI
Default: NULL
Extra:
Privileges: select,insert,update,references
Comment:
***** 2. row ****
Field: checkNumber
Type: varchar(50)
Collation: utf8_general_ci
Null: NO
Key: PRI
Default: NULL
Extra:
Privileges: select,insert,update,references
Comment:
***** 3. row ****
Field: paymentDate
Type: date
Collation: NULL
Null: NO
Key:
Default: NULL
```

列出表的列

```
Extra:  
Privileges: select,insert,update,references  
Comment:  
***** 4. row *****  
Field: amount  
Type: decimal(10,2)  
Collation: NULL  
Null: NO  
Key:  
Default: NULL  
Extra:  
Privileges: select,insert,update,references  
Comment:  
4 rows in set (0.02 sec)
```

ERROR:

```
No query specified
```

```
mysql>
```

如您所见，`SHOW FULL COLUMNS` 命令将排序规则，权限和注释列添加到结果集中。

`SHOW COLUMNS` 命令允许使用`LIKE`运算符或`WHERE`子句来过滤表的列：

```
SHOW COLUMNS FROM table_name LIKE pattern;
```

```
SHOW COLUMNS FROM table_name WHERE expression;
```

例如，要显示`payments` 表中以字母 `c` 开头的列，请使用 `LIKE` 运算符，如下所示：

列出表的列

```
mysql> SHOW COLUMNS FROM payments LIKE 'c%';
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| customerNumber | int(11) | NO | PRI | NULL |       |
| checkNumber | varchar(50) | NO | PRI | NULL |       |
+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

在本教程中，您已经学会了如何使用MySQL `SHOW COLUMNS` 命令和 `DESC` 语句来显示表的列。

本教程将向您展示如何在MySQL数据库中列出用户。

MySQL显示用户：列出所有用户

要印出所有MySQL中的所有用户，您是不是想使用MySQL SHOW USERS 命令？

不幸的是，MySQL没有类似SHOW DATABASES，SHOW TABLES等那样的 SHOW USERS 命令，因此列出MySQL数据库服务器中的所有用户，可使用以下查询：

```
SELECT  
    user  
FROM  
    mysql.user;
```

在此语句中，我们从 mysql 数据库的 user 表查询用户数据信息。

要执行此查询，您必须以管理员身份登录到MySQL数据库服务器。

```
C:\Users\Administrator> mysql -u root -p  
Enter password: *****  
mysql> use mysql;  
Database changed  
mysql> SELECT user FROM user;
```

以下显示了以上查询的输出：

```
mysql> USE mysql  
Database changed  
mysql> SELECT user FROM user;  
+-----+  
| user |  
+-----+  
| rfc  |  
| superadmin |  
| mysql.sys |  
| root |  
| super |  
+-----+  
5 rows in set (0.27 sec)
```

如上所见，在本地数据库中有五个用户。

要获取有关用户表的更多信息，可以使用以下命令预览其列：

```
DESC user;
```

例如，要显示用户和其他信息(如主机，帐户锁定和密码到期状态)，请使用以下查询：

```
SELECT
    user,
    host,
    account_locked,
    password_expired
FROM
    mysql.user;
```

执行上面查询语句，得到以下结果 -

```
mysql> SELECT
    user,
    host,
    account_locked,
    password_expired
FROM
    mysql.user;
+-----+-----+-----+-----+
| user | host | account_locked | password_expired |
+-----+-----+-----+-----+
| root | localhost | N | N |
| mysql.sys | localhost | Y | N |
| superadmin | % | N | N |
| super | localhost | N | N |
| rfc | % | N | N |
+-----+-----+-----+-----+
5 rows in set
```

显示当前用户

列出用户

要获取有关当前用户的信息，请使用 `user()` 函数，如以下语句所示：

```
mysql> SELECT user();
+-----+
| user() |
+-----+
| root@localhost |
+-----+
1 row in set
```

或者使用 `current_user()` 函数：

```
mysql> SELECT current_user();
+-----+
| current_user() |
+-----+
| root@localhost |
+-----+
1 row in set
```

在本示例的情况下，当前用户是：`root@localhost`。

显示当前登录的用户

要列出当前登录MySQL数据库服务器的所有用户，请执行以下语句：

```
SELECT
    user,
    host,
    db,
    command
FROM
    information_schema.processlist;
```

执行上面语句，得到以下结果

```
mysql> SELECT
    user,
    host,
    db,
    command
FROM
    information_schema.processlist;
+-----+-----+-----+-----+
| user | host      | db     | command |
+-----+-----+-----+-----+
| root | localhost:51897 | NULL   | Sleep   |
| root | localhost:51898 | yiibaidb | Sleep   |
| root | localhost:55313 | NULL   | Query   |
| root | localhost:52939 | mysql   | Sleep   |
+-----+-----+-----+-----+
4 rows in set
```

正如使用可以看到的，目前有四个用户登录在MySQL数据库中，一个是执行“查询”状态，其它三个是“睡眠”状态。

在本教程中，您已经学习了如何通过查询 `mysql` 数据库中的用户表中的数据来列出MySQL数据库服务器中的所有用户。