# mysql视图

## 视图简介

数据库视图是虚拟表或逻辑表，它被定义为具有[连接](http://www.yiibai.com/mysql/inner-join.html)的SQL [SELECT](http://www.yiibai.com/mysql/select-statement-query-data.html)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。

## 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数据访问。 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。 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。

## MySQL视图的实现和限制

*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](http://www.yiibai.com/mysql/temporary-table.html)，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](http://www.yiibai.com/mysql/managing-sql-views.html)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](http://www.yiibai.com/mysql/select-statement-query-data.html)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](http://www.yiibai.com/mysql/create-drop-index.html)。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](http://www.yiibai.com/mysql/subquery/)来定义视图的。

如果删除或[重命名](http://www.yiibai.com/mysql/rename-table.html)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](http://www.yiibai.com/mysql/create-sql-updatable-views.html)表中数据。基于具有[连接](http://www.yiibai.com/mysql/inner-join.html)，子查询等的复杂SELECT语句创建的视图无法更新。

MySQL不像：[Oracle](http://www.oraok.com/)，[PostgreSQL](http://www.yiibai.com/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。

## MySQL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]

SQL

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](http://www.yiibai.com/mysql/select-statement-query-data.html)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](http://www.yiibai.com/mysql/min.html)，[MAX](http://www.yiibai.com/mysql/max-function.html)，[SUM](http://www.yiibai.com/mysql/sum.html)，[COUNT](http://www.yiibai.com/mysql/count.html)，[AVG](http://www.yiibai.com/mysql/avg.html)等)或[DISTINCT](http://www.yiibai.com/mysql/distinct.html)，[GROUP BY](http://www.yiibai.com/mysql/group-by.html)，[HAVING](http://www.yiibai.com/mysql/having.html)，[LIMIT](http://www.yiibai.com/mysql/limit.html)，[UNION](http://www.yiibai.com/mysql/sql-union-mysql.html)，[UNION ALL](http://www.yiibai.com/mysql/sql-union-mysql.html)，[子查询](http://www.yiibai.com/mysql/subquery/)，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](http://www.yiibai.com/mysql/temporary-table.html)，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](http://www.yiibai.com/mysql/create-sql-updatable-views.html)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](http://www.yiibai.com/mysql/where.html)中包含[子查询](http://www.yiibai.com/mysql/subquery.html)，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;

SQL

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

SQL

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

SQL

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;

SQL

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

SQL

**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;

SQL

现在，我们可以从BigSalesOrder视图查询数据，如下所示：

SELECT

orderNumber, total

FROM

BigSalesOrder;

SQL

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

SQL

**使用连接表创建视图**

以下是使用[INNER JOIN](http://www.yiibai.com/mysql/inner-join.html)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;

SQL

要查询customerOrders视图中的数据，请使用以下查询：

SELECT \* FROM customerOrders;

SQL

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

Shell

**使用子查询创建视图**

以下说明如何使用[子查询](http://www.yiibai.com/mysql/subquery.html)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;

SQL

查询上述视图：aboveAvgProducts的数据简单如下：

SELECT

\*

FROM

aboveAvgProducts;

SQL

执行上面查询语句，得到以下结果 -

+-------------+-----------------------------------------+----------+

| productCode | productName | buyPrice |

+-------------+-----------------------------------------+----------+

| S10\_4962 | 1962 LanciaA Delta 16V | 103.42 |

| S18\_2238 | 1998 Chrysler Plymouth Prowler | 101.51 |

| S10\_1949 | 1952 Alpine Renault 1300 | 98.58 |

|\*\*\*\*\*\*\*\*\*\*\*\*\* 此处省略了一大波数据 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|

| 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

SQL

在本教程中，我们向您展示了如何使用CREATE VIEW语句创建视图。

## 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](http://www.yiibai.com/mysql/insert-statement.html)或[UPDATE](http://www.yiibai.com/mysql/update-data.html)语句通过可更新视图插入或更新基表的行。 另外，您可以使用[DELETE](http://www.yiibai.com/mysql/delete-statement.html)语句通过视图删除底层表的行。

但是，要创建可更新[视图](http://www.yiibai.com/mysql/introduction-sql-views.html)，定义视图的[SELECT语句](http://www.yiibai.com/mysql/select-statement-query-data.html)不能包含以下任何元素：

* [聚合函数](http://www.yiibai.com/mysql/aggregate-functions.html)，如：[MIN](http://www.yiibai.com/mysql/min.html)，[MAX](http://www.yiibai.com/mysql/max-function.html)，[SUM](http://www.yiibai.com/mysql/sum.html)，[AVG](http://www.yiibai.com/mysql/avg.html)，[COUNT](http://www.yiibai.com/mysql/count.html)等。
* [DISTINCT子句](http://www.yiibai.com/mysql/distinct.html)
* [GROUP BY子句](http://www.yiibai.com/mysql/group-by.html)
* [HAVING子句](http://www.yiibai.com/mysql/having.html)
* [UNION](http://www.yiibai.com/mysql/sql-union-mysql.html)或UNION ALL子句
* [左连接](http://www.yiibai.com/mysql/left-join.html)或外连接。
* SELECT子句中的[子查询](http://www.yiibai.com/mysql/subquery.html)或引用该表的[WHERE子句](http://www.yiibai.com/mysql/where.html)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](http://www.yiibai.com/mysql/create-sql-views-mysql.html)创建视图，则无法更新视图。

请注意，有时可以使用[内部连接](http://www.yiibai.com/mysql/inner-join.html)创建基于多个表的可更新视图。

## 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];

SQL

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。

## 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]

SQL

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;

SQL

要验证更改，可以从organization视图中查询数据：

SELECT

\*

FROM

Organization;

SQL

执行上面查询语句，得到以下结果 -

Shell

**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;

## 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]

SQL

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;

SQL

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。

# 存储过程

## MySQL存储过程简介及优缺点

### 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](http://www.yiibai.com/mysql/triggers.html)，其他存储过程以及[Java](http://www.yiibai.com/jdbc/%22Java%22)，[Python](http://www.yiibai.com/python/)，[PHP](http://www.yiibai.com/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](http://www.yiibai.com/mysql/stored-function.html)，[触发器](http://www.yiibai.com/mysql/triggers.html)和[事件](http://www.yiibai.com/mysql/triggers/working-mysql-scheduled-event.html)。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。

### 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当于逻辑运算。
* 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并不容易。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。

## MySQL存储过程入门

### 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](http://www.yiibai.com/introduction-to-sql-stored-procedures.html)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 ;

SQL

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](http://www.yiibai.com/mysql/select-statement-query-data.html)来查询products表中的数据。

### 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();

SQL

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();

## MySQL存储过程的变量

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](http://www.yiibai.com/mysql/stored-procedure.html)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;

SQL

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](http://www.yiibai.com/mysql/data-types.html)，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;

SQL

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;

SQL

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;

SQL

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

SQL

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)](http://www.yiibai.com/mysql/sample-database.html" \o "示例数据库(yiibaidb)" \t "_blank)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。

## MySQL存储过程参数

### 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)

SQL

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](http://www.yiibai.com/variables-in-stored-procedures.html)一样，参数的数据类型可以是任何有效的[MySQL数据类型](http://www.yiibai.com/mysql/data-types.html)。

如果存储过程有多个参数，则每个参数由逗号(,)分隔。

### 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 `yiibaidb`;

DROP procedure IF EXISTS `GetOfficeByCountry`;

DELIMITER $$

USE `yiibaidb`$$

CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))

BEGIN

SELECT \*

FROM offices

WHERE country = countryName;

END$$

DELIMITER ;

SQL

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');

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 `yiibaidb`;

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 ;

SQL

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](https://github.com/total" \o "@total" \t "_blank))以获取返回值。

CALL CountOrderByStatus('Shipped',@total);

SELECT @total;

SQL

执行上面查询语句后，得到以下结果 -

+--------+

| @total |

+--------+

| 303 |

+--------+

1 row in set

SQL

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

SQL

### 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 ;

SQL

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

SQL

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。

## MySQL存储过程返回多个值

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。

### 返回多个值的存储过程示例

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

SQL

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

SQL

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](http://www.yiibai.com/mysql/count.html)的[SELECT语句](http://www.yiibai.com/mysql/select-statement-query-data.html)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

## if语句

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](http://www.yiibai.com/mysql/variables-in-stored-procedures.html)，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。

### IF语句语法

下面说明了IF语句的语法：

IF expression THEN

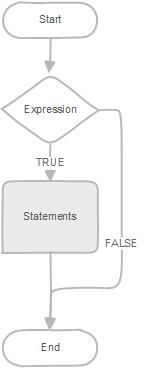
statements;

END IF;

SQL

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;

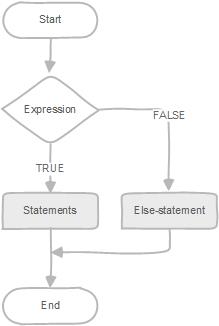
ELSE

else-statements;

END IF;

SQL

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;

...

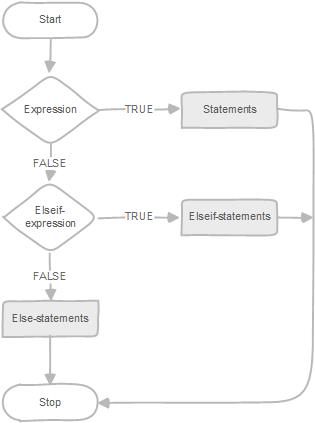
ELSE

else-statements;

END IF;

SQL

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';

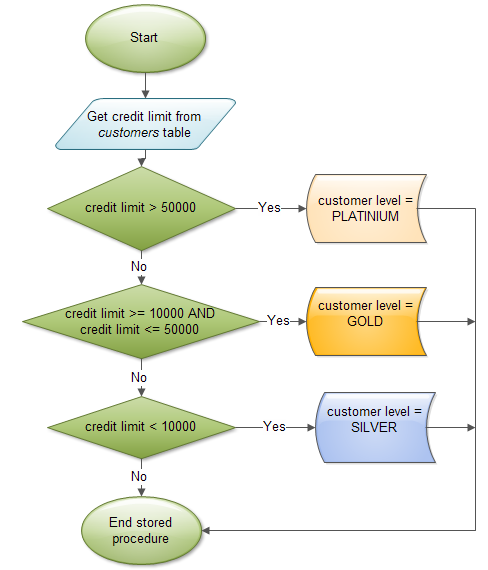
ELSEIF creditlim < 10000 THEN

SET p\_customerLevel = 'SILVER';

END IF;

END$$

以下流程图演示了确定客户级别的逻辑 -



## CASE语句

除了[IF语句](http://www.yiibai.com/mysql/if-statement.html)，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;

SQL

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\_shiping varchar(50))

BEGIN

DECLARE customerCountry varchar(50);

SELECT country INTO customerCountry

FROM customers

WHERE customerNumber = p\_customerNumber;

CASE customerCountry

WHEN 'USA' THEN

SET p\_shiping = '2-day Shipping';

WHEN 'Canada' THEN

SET p\_shiping = '3-day Shipping';

ELSE

SET p\_shiping = '5-day Shipping';

END CASE;

END$$

SQL

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](http://www.yiibai.com/mysql/stored-procedures-parameters.html)，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;
* SQL
* 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';

SQL

执行上面查询语句，得到以下结果 -

+----------------+

| Customer Level |

+----------------+

| PLATINUM |

+----------------+

1 row in set

## MySQL使用IF和CASE语句的技巧

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](http://www.yiibai.com/mysql/case-statement.html)比[IF语句](http://www.yiibai.com/mysql/if-statement.html)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](http://www.yiibai.com/mysql/error-handling-in-stored-procedures.html)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。

## MySQL存储过程循环

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

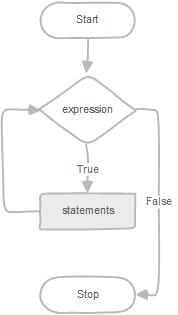
statements

END WHILE

SQL

WHILE循环在每次迭代开始时检查表达式。 如果expressionevaluates为TRUE，MySQL将执行WHILE和END WHILE之间的语句，直到expressionevaluates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](http://www.yiibai.com/mysql/variables-in-stored-procedures.html)的值大于5。
* 然后，使用[SELECT语句](http://www.yiibai.com/mysql/select-statement-query-data.html)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\_loopstored调用存储过程：

CALL test\_mysql\_while\_loop();

SQL

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

SQL

### REPEAT循环

REPEAT循环语句的语法如下：

REPEAT

statements;

UNTIL expression

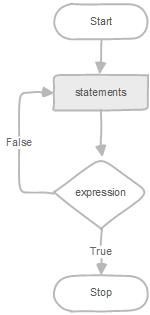
END REPEAT

SQL

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](http://www.yiibai.com/php/)，C/C++，[Java](http://www.yiibai.com/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

## MySQL游标

### MySQL游标简介

要处理[存储过程](http://www.yiibai.com/mysql/stored-procedure.html)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](http://www.yiibai.com/mysql/select-statement-query-data.html)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](http://www.yiibai.com/mysql/stored-procedure.html)，[存储函数](http://www.yiibai.com/mysql/stored-function.html)和[触发器](http://www.yiibai.com/mysql/triggers.html)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;

SQL

游标声明必须在[变量](http://www.yiibai.com/mysql/variables-in-stored-procedures.html)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语句初始化游标的结果集，因此您必须在从结果集中提取行之前调用OPEN语句。

OPEN cursor\_name;

SQL

然后，使用FETCH语句来检索光标指向的下一行，并将光标移动到结果集中的下一行。

FETCH cursor\_name INTO variables list;

SQL

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;

SQL

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。

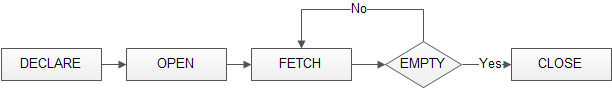
要声明一个NOT FOUND处理程序，参考以下语法：

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

SQL

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employees表中所有员工的电子邮件列表。

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;

SQL

接下来，使用OPEN语句打开email\_cursor：

OPEN email\_cursor;

SQL

然后，迭代电子邮件列表，并使用分隔符(;)[连接](http://www.yiibai.com/mysql/sql-concat-in-mysql.html)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;

SQL

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;

SQL

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 ;

SQL

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;

## MySQL存储过程错误处理

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;

SQL

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;

SQL

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;

SQL

以下处理程序如果没有更多的行要提取，在[光标](http://www.yiibai.com/mysql/cursor.html)或[SELECT INTO](http://www.yiibai.com/mysql/select-statement-query-data.html)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;

SQL

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](http://www.yiibai.com/mysql/create-table.html)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)

);

SQL

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](http://www.yiibai.com/mysql/foreign-key.html)。

接下来，创建一个[存储过程](http://www.yiibai.com/mysql/getting-started-with-mysql-stored-procedures.html)，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 ;

SQL

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);

SQL

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);

SQL

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 |

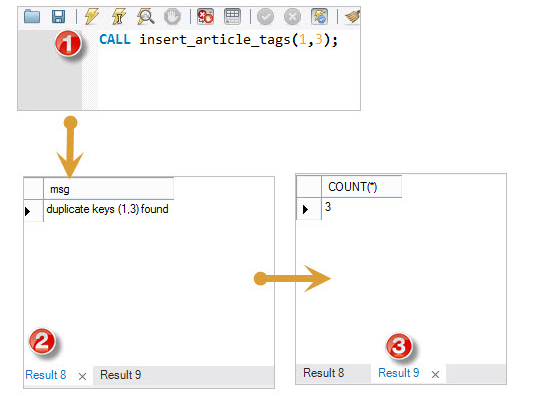
+----------+

1 row in set

Query OK, 0 rows affected

SQL

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 ;

SQL

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 |

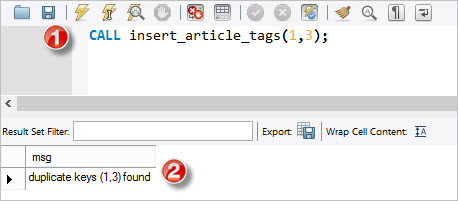
+-------------------------------+

| 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。 SQLEXCPETION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 ;

SQL

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);

SQL

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;

SQL

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;

SQL

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;

# MySQL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](http://www.yiibai.com/mysql/stored-procedure.html)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

SQL

*首先*，在CREATE FUNCTION子句之后指定存储函数的名称。  
*其次*，列出括号内存储函数的所有[*参数*](http://www.yiibai.com/mysql/stored-procedures-parameters.html)。 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](http://www.yiibai.com/mysql/data-types.html)。  
*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](http://www.yiibai.com/mysql/if-statement.html)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](http://www.yiibai.com/mysql/select-statement-query-data.html)中调用CustomerLevel()存储函数，如下所示：

SELECT

customerName, CustomerLevel(creditLimit)

FROM

customers

ORDER BY customerName;

下面，来重写在[MySQL IF语句教程](http://www.yiibai.com/mysql/if-statement.html)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。