第 5 章 数据库中的对象

数据库中的基本对象是表。但是除了表以外，数据库中还有很多其它的对象，例如索引、视图、存储过程、触发器、游标、函数等。这些数据库对象对于提高数据查询的效率、提升数据的安全性与完整性以及实现数据操作代码的共享性和数据操作的灵活性等方面起到了良好的辅助作用。本章详细介绍了数据库中常用对象的概念、作用和使用方法。重点的对象是索引、视图、存储过程和触发器。

5.1 索引

5.1.1 索引概述

在日常生活中我们会经常使用索引，例如图书的目录、词典的检索表等。借助索引，人们能够很快地找到需要的东西。如果把表比作一本书，则表的索引就如书的目录一样，通过索引可大大提高数据的查询速度。

索引是数据库随机检索的常用手段，它实际上就是记录的关键字与其相应地址的对应表。例如，当我们要在一本数据库原理教材中查找有关“函数依赖”的内容时，应该先通过目录找到“函数依赖”所对应的页码，然后从该页中找出需要的信息。这种方法比直接翻阅书的内容要快。

索引之所以能加快查询速度，是因为索引文件只有两个字段。一个是排序后的的索引字段，用于快速查找；另一个是该记录在数据表中的记录号，用于定位读取。这样索引文件的大小就要比数据表小得多，并且使用二分法对索引字段进行查找，从而比无索引时的顺序查找速度要快得多。

索引的作用可归纳为：

(1) 可以加快数据的检索速度。

(2) 唯一索引可以保证数据记录的唯一性。

(3) 在使用排序和分组进行数据查询时，可以显著减少查询中排序和分组的时间。

(4) 在进行连接查询时可以加快表与表之间的连接。这在实现数据的参照完整性方面有特别的意义。

建立索引是加快表的查询速度的有效手段。SOL 语句支持用户根据应用环境的需要在基本表上建立一个或多个索引，以提供多种存取路径。一般说来，建立与删除索引由数据库管理员或表的属主负责完成。DBMS 在存取数据时会自动选择合适的索引作

为存取路径，用户不必也不能人为选择索引。

虽然索引可以提高数据的查询效率，但它会占用一定的存储空间，并且为了维护索引的有效性，在向表中插入、删除或者更新数据时，数据库还要自动执行额外的操作来维护索引。随着数据量的增加，创建和维护索引所消耗的时间也会随之而增加。因此，使用索引时，应综合考虑它的优缺点来科学地设计，才能提高数据库的性能。

5.1.2 索引的分类

MYSQL 中的索引种类比较多。按照索引字段的数量、类型和作用等方式划分，主要包括以下 6 种。

（1）普通索引

普通索引是 MySQL 中的基本索引类型。它允许索引字段有重复值和空值，由关键字 KEY 或 INDEX 定义，可以创建在任何数据类型中。普通索引的唯一任务是加快对数据的访问速度。

（2）唯一索引

唯一索引由关键字 UNIQUE 定义，要求索引字段的值不能重复，但允许为空值。创建唯一索引的目的往往不是为了提高访问速度，而只是为了避免数据出现重复。另外 MySQL 中的主键索引是一种特殊的唯一索引，它不允许有空值。

（3）全文索引

全文索引是由 FULLTEXT 定义的索引，是指在定义索引的字段上支持值的全文查找。该索引类型允许在索引字段上插入重复值和空值，它只能创建在 CHAR、VARCHAR 或 TEXT 等字符类型的字段上。

（4）空间索引

空间索引是由 SPATIAL 定义的索引，是只能在 GEOMETRY、POINT、LINESTRING 和 POLYGON 等空间数据类型的字段上建立的索引。需要注意的是，创建空间索引的字段，必须将其声明为 NOT NULL。

（5）单列索引

单列索引指在表中单个字段上创建的索引。它可以是普通索引、唯一索引或者全文索引，只要保证该索引关键字为表中的一个字段即可。

（6）多列索引

多列索引指在表中多个字段上创建的索引。只有在查询条件中使用了这些字段中的第一个字段时，该索引才会被使用。例如已在“成绩表”上建立了以“学号”和“课程

编号”两个字段为索引字段的多列索引，那么只有在查询条件中使用了“学号”字段时，该索引才会被使用。

5.1.3 索引的设计原则

索引设计不合理或缺少索引都会给数据库的应用造成障碍。高效的索引对于用户获得良好的性能体验非常重要。设计索引时，应该考虑以下原则。

（1）索引并非越多越好

一个表中如有大量的索引，不仅占用磁盘空间，而且会影响 INSERT、 UPDATE、DELETE 等语句的性能。因为在更改表中的数据的同时，索引也会被 DBMS 自动地进行调整和更新。对经常查询的字段应该建立索引，但要避免对不必要的字段建立索引。

（2）避免对经常更新的表建立过多的索引

需要经常更新数据的表应该避免建立过多的索引，并且索引中的字段要尽可能少，这样系统消耗在索引维护上的代价才小。

（3）数据量小的表不建议使用索引

当数据量较少时，花费在查询上的时间可能比遍历索引的时间还要短，索引可能不会产生优化的效果。

（4）在取值重复率较大的字段上不要建立索引

例如“教师表”中的“性别”字段，其取值基本只有“男”和“女”两个值，这样的字段就无须建立索引。建立索引后不但不会提高查询效率，反而会严重降低更新速度。

（5）为经常需要进行排序、分组和连接查询的字段建立索引

在关系数据中进行排序、分组和连接查询时，需要进行大量的查找比较运算，所以应该为频繁进行排序或分组的字段和经常进行连接查询的字段创建索引。

5.1.4 创建索引

在 MySQL 中创建索引有三种方式。

1. 在使用 CREATE TABLE 语句创建表时创建索引

基本语法格式为：

```
CREATE TABLE  表名
（字段名  数据类型  [完整性约束条件],
  字段名  数据类型  [完整性约束条件],
          ……
  字段名  数据类型
```

```
    [UNIQUE | FULLTEXT | SPATIAL]
    INDEX|KEY [别名](字段名[(长度)]) [ASC|DESC]
);
```

其中与索引有关的参数说明如下：

- UNIQUE：可选参数，表示创建唯一索引。
- FULLTEXT：可选参数，表示创建全文索引。
- SPATIAL：可选参数，表示创建空间索引
- INDEX 和 KEY：创建的索引的关键词，二者任选其一使用即可。
- 别名：可选参数，用于给索引另取一个名称。如果不用别名，则使用索引字段的名称作为索引名。
- 字段名：指定索引对应字段的名称。
- 长度：可选参数，用于指定字段中用于创建索引的长度，默认为整个字段值。
- ASC 和 DESC：可选参数，指定索引字段的排序方式。其中 ASC 表升序，DESC 表示降序排列。默认为升序排列。

【例 5-1】创建 S 表，并以 sname 字段建立普通索引。

```
CREATE TABLE S
(    sno     CHAR(10) ,
     sname     VARCHAR(20),
     age     INT,
     sex    CHAR(2),
     dept    VARCHAR(20),
     INDEX (sname)
);
```

【例 5-2】创建 SC 表，并以 sno 和 cno 两个字段组合建立名为 uk_sno_cno 多列唯一索引。

```
CREATE TABLE SC
(    sno     CHAR(10) ,
     cno     CHAR(15),
     score    decimal(4,1),
     UNIQUE INDEX uk_sno_cno(sno,cno)
);
```

2. 使用 CREATE INDEX 语句在已经存在的表上创建索引

基本语法格式为：

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX  索引名
ON  表名(字段名[(长度)]) [ASC|DESC] [,…]);
```

【例 5-3】为 SC 表建立名为 idx_sno 的按 sno 字段降序排列的普通索引。

```
CREATE INDEX idx_sno ON SC(sno DESC);
```

【例 5-4】以 sno 字段为 S 表建立名为 uk_sno 的唯一索引。

```
CREATE UNIQUE INDEX uk_sno ON S(sno);
```

3. 使用 ALTER TABLE 语句在已经存在的表上创建索引

基本语法格式为：

```
ALTER TABLE  表名  ADD [UNIQUE|FULLTEXT|SPATIAL] INDEX
索引名  (字段名[(长度)] [ASC|DESC]);
```

这种方式创建索引与使用 CREATE INDEX 语句的作用完全一样。

【例 5-5】与例 5-3 同等效果的 ALTER TABLE 语句如下：

```
ALTER TABLE SC ADD INDEX idx_sno (sno DESC);
```

【例 5-6】与例 5-4 同等效果的 ALTER TABLE 语句如下：

```
ALTER TABLE S ADD UNIQUE INDEX uk_sno (sno);
```

5.1.5 查看索引

在 MySQL 中使用 SHOW CREATE TABLE 语句可以查看表的结构，同时也能查看表中存在哪些索引。如图 5-1 所示，表 S 中已经存在一个名为 idx_sname 的普通索引和一个名为 uk_sno 的唯一索引。

```
mysql> SHOW CREATE TABLE S \G
*************************** 1. row ***************************
       Table: S
Create Table: CREATE TABLE `s` (
  `sno` char(10) DEFAULT NULL,
  `sname` varchar(20) DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `sex` char(2) DEFAULT NULL,
  `dept` varchar(20) DEFAULT NULL,
  UNIQUE KEY `uk_sno` (`sno`),
  KEY `idx_sname` (`sname`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

图 5-1 查看表的索引

使用 EXLAIN 语句可以查看在执行 SQL 查询时索引的使用情况。从图 5-2 所示的

结果中可以看出，第一条查询命令使用了 uk_sno_cno 索引，而第二条查询命令的 key 为 NULL，表示查询时没有使用任何索引。

```
mysql> EXPLAIN  select * from SC where Sno='s001';
+----+-------------+-------+------+---------------+------------+---------+-------+------+-------------+
| id | select_type | table | type | possible_keys | key        | key_len | ref   | rows | Extra       |
+----+-------------+-------+------+---------------+------------+---------+-------+------+-------------+
|  1 | SIMPLE      | SC    | ref  | uk_sno_cno    | uk_sno_cno | 31      | const |    1 | Using where |
+----+-------------+-------+------+---------------+------------+---------+-------+------+-------------+
1 row in set (0.00 sec)

mysql> EXPLAIN  select * from SC where cno='c012';
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | SC    | ALL  | NULL          | NULL | NULL    | NULL |    1 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

图 5-2 查看索引的使用

5.1.6 删除索引

由于索引会占用一定的系统资源，因此，为了避免影响数据库性能，应该及时删除不再使用的索引。MySQL 中有两种方法删除索引。

1. 使用 DROP INDEX 删除索引

基本语法格式如下：

```
DROP INDEX  索引名  ON  表名;
```

【例 5-7】删除表 SC 中名称为 idx_sname 的索引。

```
DROP INDEX idx_sname ON SC;
```

2. 使用 ALTER TABLE 语句删除索引

基本语法格式如下：

```
ALTER TABLE  表名  DROP INDEX  索引名;
```

【例 5-8】删除表 SC 中名称为 uk_sno_cno 的索引。

```
ALTER TABLE SC DROP INDEX uk_sno_cno;
```

5.2 视图

5.2.1 视图的基本概念

视图是从一个或几个基表（或视图）中导出的表，是一种虚表。用于产生视图的表叫做该视图的基表。数据库中仅存放了视图的定义，不存放视图对应的数据。通过视图看到的数据实际只是执行视图中定义的数据查询命令而查询出的存放在基表中的数据。

当基表中的数据发生变化，视图中的数据也就会随之改变。这使得视图就像一个窗体，透过它能够看到数据库中感兴趣的那部分数据及其变化。

可以像数据库中的任何其它的数据表一样查询和更新视图。当通过视图更新数据（INSERT、DELETE、UPDATE）时，实际上是在改变其基表中的数据；相反地，基表数据的改变也会自动反映在由基表产生的视图中。不过要注意的是因为并非所有的视图更新都能转换成可行的基表更新，所以数据库管理系统对视图的更新操作往往存在一定的限制。

视图通常用来实现以下三种功能：

- 筛选出基表中的频繁操作的数据供用户简单地按视图名访问。
- 只提供必要的数据视图，防止未经许可的用户访问敏感数据。
- 将多个物理数据表抽象为一个逻辑数据表。

视图的主要功能包括：

(1) 视图能够简化用户的操作。视图使用户可以将注意力集中在自己关心的数据上，如果这些数据不是直接来自于基表，则可以通过定义视图，使得用户眼中的数据结构简单、清晰，并且可以简化用户的数据查询操作。例如，那些来源于若干张表连接查询的视图，就将表与表之间的连接操作对用户隐藏了起来。换句话说，用户所做的只是对一个虚表的简单查询，而这个虚表是怎样得到的，用户无需了解。

(2) 视图使用户能从多种角度看待同一数据。视图机制使不同的用户能以不同的方式看待同一组数据。当不同用户使用同一个数据库时，这种灵活性是非常重要的。

(3) 视图能够实现数据库的逻辑独立性。数据的逻辑独立性是指当数据库的逻辑结构改变时（如给基表增加了新的字段），用户和应用程序可以不受影响地继续使用数据。

(4) 视图能够对机密数据提供安全保护。有了视图机制，就可以在设计数据库应用系统时，对不同的用户定义不同的视图，并设置不同的权限，使机密数据不出现在不应看到这些数据的用户视图上。

5.2.2 定义视图

在 MSQL 中，创建视图的命令的基本语法格式为：

```
CREATE  [ OR REPLACE]  VIEW 视图名[(字段名列表)]
AS
SELECT 语句
[ WITH  CHECK OPTION ]
```

其中：

（1）OR REPLACE：若存在同名的视图时，原视图将被新创建的视图覆盖。该子句使得新建视图命令同时也具备了修改视图的功能。

（2）字段名列表：指定视图查询结果的字段名，如果没有此选项，视图查询结果的字段名和 SELECT 子句中的字段名一致。

（3）WITH CHECK OPTION：使用该子句后，在更新视图时对新插入或修改的数据进行检查，确保视图数据的更新要使得视图定义里的查询语句中的 WHERE 子句的结果为“真”。

【例 5-9】在 college 数据库中创建一个基于 teacher 表的信息管理系教师信息的视图，视图名为 t_inf_view。

```
USE college;
CREATE VIEW t_inf_view
AS
SELECT * FROM teacher WHERE Dep='信息管理';
```

执行该语句后，使用“show tables”命令即可看到 college 数据库中新增了一个名为 t_inf_view 的“表”。接下来使用 SELECT 语句查询 t_inf_view 视图，则会显示出所有信息管理系老师的工号、姓名、性别等个人完整信息，其数据源自基表 teacher。如图 5-3 所示。

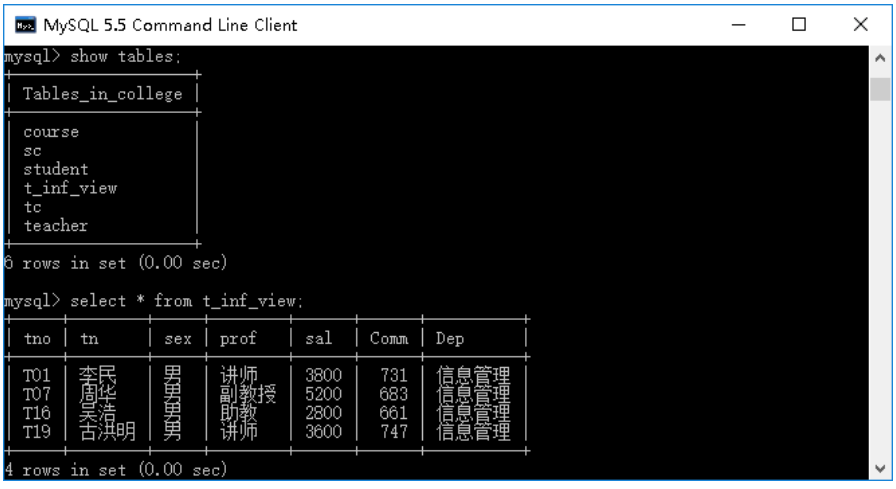

图 5-3 查看创建 t_inf_view 视图的结果

【例 5-10】在 college 数据库中创建一个基于 teacher 表的视图 t_view，要求视图中只有教师的姓名（tn）、性别（sex）、职称（prof）和系别（Dep），且字段名为中文名称。

```
USE college;
CREATE VIEW t_view(姓名,性别,职称,系别)
AS
SELECT tn, sex, prof,Dep
FROM teacher;
```

由于 SELECT 命令可以给查询结果的字段换名，所以也可以用以下的命令实现：

```
USE college;
CREATE VIEW t_view
AS
SELECT tn as  姓名, sex as  性别, prof as  职称,Dep as  系别
FROM teacher;
```

图 5-4 显示了使用 SELECT 语句查询 t_view 视图的结果。

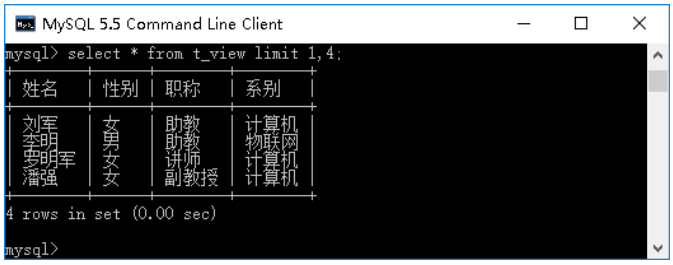

图 5-4 查 t_view 视图的结果

【例 5-11】在 college 数据库中，基于例 5-9 创建的视图 t_inf_view 再创建一个信息管理系讲师的 姓名（tn）、职称（prof）和系别(Dep)的视图 t_inf_lec_view。

```
USE college;
CREATE VIEW t_inf_lec_view
AS
SELECT tn,sex,prof
FROM t_inf_view
WHERE prof='讲师';
```

图 5-5 显示了使用 SELECT 语句查询 t_inf_lec_view 视图的结果。

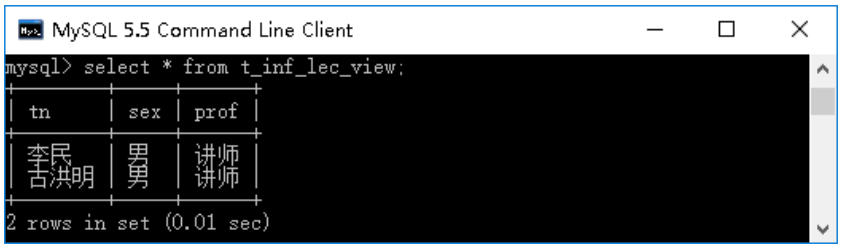

图 5-5 查询 t_inf_lec_view 视图的结果

5.2.3 查看视图

查看视图是指查看数据库中已经存在的视图的定义。MySQL 中专门用于查看视图的命令是 SHOW CREATE VIEW 语句，其基本语法格式为：

```
SHOW CREATE VIEW 视图名;
```

【例 5-12】使用 SHOW CREATE VIEW 查看视图 t_view。

```
SHOW CREATE VIEW t_ view;
```

执行结果如图 5-6 所示。从图中可以清楚地看到视图的基表、结构及字符编码等详细信息。

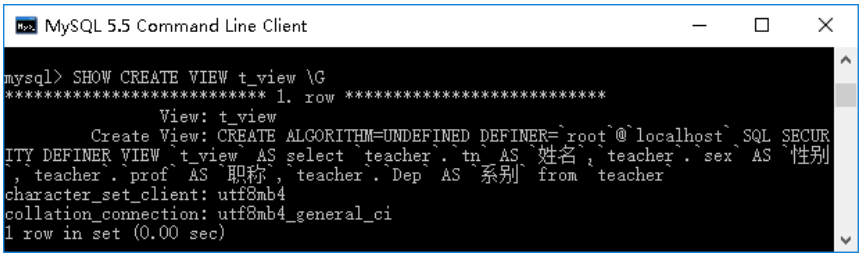

图 5-6 SHOW CREATE VIEW 命令示例

由于视图是一种虚表，在很多场合下都可以当作表来使用，所以在 MySQL 中大多数跟表的有关的语句也能作用于视图。例如前面示例中使用过的“show tables”命令就能查看数据库已经存在的表名和视图名。类似的命令还有 DESCRIBE、SHOW TABLE STATUS 等。

【例 5-13】用 DESCRIBE 命令查看视图 t_view 的基本信息

```
DECSRIBE t_view ;
```

执行结果如图 5-7 所示，显示出了 t_view 视图的所有字段的字段名（Field）、数据类型（Type）、是否允许取空（Null）、是否有索引（Key）等基本信息。

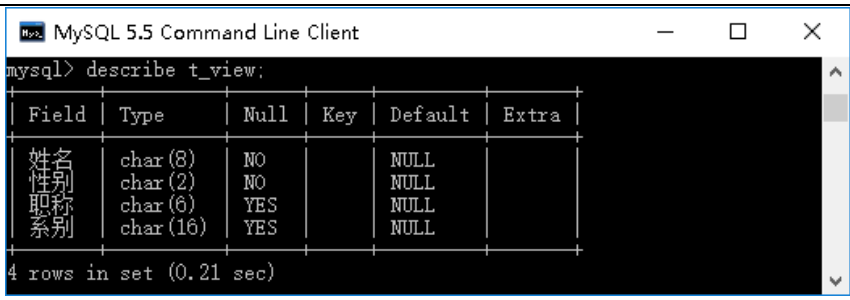

图 5-7 查看 t_ view 视图字段信息

5.2.4 修改视图

修改视图就是修改数据库中已经存在的视图的定义，本质上是对已有视图的重新定义。在 MySQL 中修改视图主要使用 ALTER VIEW 语句，其基本语法格式为：

```
ALTER VIEW 视图名[（字段名列表）]
AS
SELECT 语句
[WITH   CHECK OPTION]
```

【例 5-14】使用 ALTER VIEW 语句，修改例 5-10 创建的 t_ view 视图，使得 t_view 视图增加一个工号(tno)列。

```
ALTER VIEW t_view(工号,姓名,性别,职称,系别)
AS
SELECT tno, tn, sex, prof, Dep
FROM teacher;
```

另外，前面已经介绍过的 MySQL 的创建视图的命令 CREATE VIEW 如果加上 OR REPLACE 短语 也能起到同样的修改视图的作用：

```
CREATE OR REPLACE VIEW t_view(工号,姓名,性别,职称,系别)
AS
SELECT tno, tn, sex, prof, Dep
FROM teacher;
```

5.2.5 删除视图

当视图不再需要时，可以将其从数据库中删除。删除视图只是删除了视图的定义，并不会影响基表中的数据。

在 MySQL 中删除视图的命令的基本语法格式为：

```
DROP VIEW [IF EXISTS］视图名列表;
```

如果要一次删除多个视图，则视图名列表中的各视图名之间用逗号分隔。IF EXISTS可选项表示如果存在被删除的视图则将其删除，如果不存在则删除操作不用抛出找不到视图的错误。

【例 5-15】删除例 5-11 创建的 t_inf_lec_view 视图。

```
DROP VIEW t_inf_lec_view;
```

接下来使用 show tables 命令即可查看到 t_inf_lec_view 已经不存在了。

5.2.6 更新视图

更新视图是指通过视图来插入（INSERT）、删除（DELETE）和更新（UPDATE）数据。由于视图是一张虚表，其中并没有数据，所以对视图的更新最终会转换成对基表数据的更新。但并非所有的视图都是可更新的。只有那些作用于视图的更新命令能够被数据库管理系统自动变换成对基表更新的正确命令的视图，才能够被更新。

【例 5-16】通过例 5-9 创建的视图 t_inf_view 给信息管理系新增一位名叫“陈丽萍”的女老师，工号为“T21”，职称为“教授”，工资 6500，补助 900。

```
INSERT INTO t_inf_view
VALUES('T21','陈丽萍','女','教授',6500,900,'信息管理');
```

命令执行成功后，通过 SELECT 命令查询 t_inf_view 视图和 teacher 表，都能发现新增了一行完整的“陈丽萍”老师的数据。

【例 5-17】通过例 5-10 创建的视图 t_view，将李民老师的职称改为副教授。

```
UPDATE t_view
SET  职称  = '副教授'
WHERE  姓名='李民';
```

命令执行成功后，通过 SELECT 命令查询 t_view 视图和 teacher 表，都能看到李民老师的职称已经修改成了副教授。

【例 5-18】利用视图 t_inf_view 将例 5-16 中新增加的“陈丽萍”老师删除。

```
DELETE from t_inf_view
WHERE  姓名='陈丽萍';
```

命令执行成功后，通过 SELECT 命令查询 t_inf_view 视图和 teacher 表，会发现无论是视图中还是基表中已经没有了陈丽萍老师的数据行。

【例 5-19】在 college 数据库中利用 teacher 表创建一个名为 t_profcnt_view 的视图，

视图的数据为分组统计出的不同职称的人数。

```
CREATE VIEW t_profcnt_view
AS
SELECT prof , count(prof) as amount
FROM teacher
GROUP BY prof;
```

命令执行成功后，通过 SELECT 命令查询 t_profcnt_view 视图可以看到视图中的统计数据。此时如果想对该视图进行更新是不允许的，因为相应的更新命令无法被准确无误地变换成对基表数据的更新。图 5-8 显示了对 t_profcnt_view 视图进行数据删除时的错误情况，因为 teacher 表中并没有任何直接与统计结果相对应的数据。

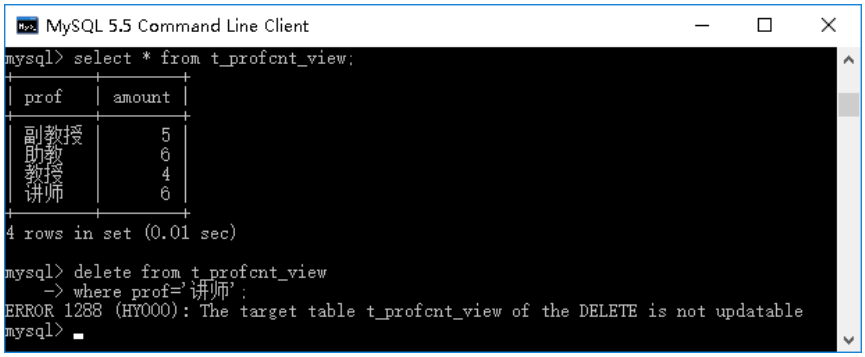

图 5-8 不可更新的视图示例

【例 5-20】根据例 5-9 创建的信息管理系老师的视图 t_inf_view，新增一位计算机系的名叫“王铭”的男老师，工号为 T22，职称为“助教”，工资 3500，补助 450。

```
INSERT INTO t_inf_view
VALUES('T22','王铭','男','助教',3500,450,'计算机');
```

该命令会被成功地执行，并且通过 SELECT 命令查询 teacher 表发现增加了王铭老师的数据。但这样的“成功”操作却产生了一个莫名其妙的逻辑：计算机系的老师可以通过信息管理系老师的视图进行更新！而产生这种结果的原因就是 t_inf_view 视图的定义中并没有要求对数据更新进行 WHERE 条件的检查。

通过带 WITH CHECK OPTION 短语的 ALTER VIEW 命令修改 t_inf_view 视图的定义，使其在更新时必须检查数据是否满足视图中的 SELECT 语句的 WHERE 条件：

```
ALTER VIEW t_inf_view
AS
SELECT * FROM teacher WHERE Dep='信息管理'
```

```
WITH CHECK OPTION;
```

如图 5-9 所示，在 t_inf_view 视图定义修改完成以后，首先试图新增一位计算机系的老师，由于不满足“WHERE Dep='信息管理'”这个条件，所以系统报错，不能成功增加。接下来如果将新增的老师的系别改为“信息管理”，则因满足 WHERE 条件而被检查通过，成功增加。

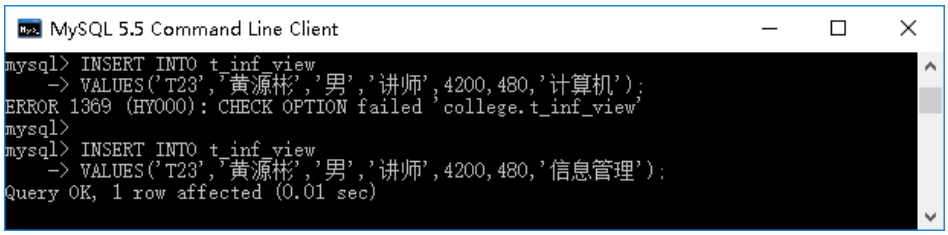

图 5-9 WITH CHECK OPTION 作用测试

5.3 存储过程

5.3.1 存储过程概述

存储过程(Stored Procedure)类似于高级语言中的子程序或函数，是数据库服务器上一组预先编译好的 SQL 语句的集合，作为一个对象存储在数据库中，可以被客户端管理工具、应用程序和其他存储过程作为一个整体来调用。在调用过程中，存储过程可以从调用者那里接收必要的输入参数，按照事先编制的程序逻辑执行存储过程中的语句序列后，向调用者返回处理结果。

存储过程增强了 SQL 编程的灵活性、语句复用性和安全性，提高了数据库应用程序的开发效率和运行效率，同时也使 SQL 程序维护起来更加容易，从而大大减少数据库开发人员的工作量，缩短整个数据库程序的开发周期。

存储过程的作用主要有：

(1) 存储过程提高了程序设计的灵活性。存储过程可以使用流程控制语句组织程序结构，方便实现结构较复杂的程序的编写，使设计过程具有很强的灵活性。

(2) 存储过程实现了程序的模块化。存储过程作为一个整体被创建后，可以被其他程序多次反复调用。对于数据库设计人员，可以根据实际情况，对存储过程进行维护，不会对调用程序产生不必要的影响。

(3) 存储过程有利于提高程序的执行速度。存储过程中包含的大量 SQL 代码或者要被反复执行的代码段在执行之前已经被预编译，所以不会象批处理那样在每次运行之

前都要进行编译，从而提高了这些代码段的执行速度。

(4) 使用存储过程能减少网络访问的负荷。在访问网络数据库的过程中，当调用存储过程时，仅需在网络中传输调用语句及其必要的参数即可，而不需要传输大量的语句，从而大大减少了网络的流量和负载。

(5) 存储过程可被作为一种安全机制来充分利用。系统管理员可以充分利用存储过程对相应数据的访问权限进行限制，从而避免非授权用户的非法访问，进一步保证数据访问的安全性。

5.3.2 创建和执行存储过程

创建存储过程的一般语法格式为：

```
CREATE PRCEDURE  存储过程名([参数定义[,…]])
   [存储过程选项]
BEGIN
   语句序列;
END;
```

说明：

(1) 参数用于在存储过程和它的调用者之间建立起一个信息传递的桥梁，使得存储过程可以从外界接收消息或者将处理的结果、状态传递出来。如果存储过程需要参数，则每一个参数都需要定义输入输出类型、名称和数据类型，格式为：

[IN | OUT | INOUT] 参数名 数据类型

其中 IN 表示在调用时需要向存储过程内传入数据的参数；OUT 表示存储过程执行完成后需要带出数据的参数；如果需要用一个参数在调用时传入数据而在执行完成后还是用它带出数据，则为 INOUT 参数。

(2) 存储过程选项的格式一般为：

LANGUAGE SQL | [NOT] DETERMINISTIC | { CONTAINS SOL| NO SQL |READS SQL DATA | MODIFIES SQL DATA} | SQL SECURITY {DEFINER| INVOKER } | COMMENT ‘string’

其中 CONTAINS SQL 表示存储程序不包含读或写数据的语句：NO SQL 表示存储程序不包含 SQL 语句； READS SQL DATA 表示存储过程包含读数据的语句，但不包含写数据的语句；MODIFIES SQL DATA 表示存储过程包含写数据的语句；SQL SECURITY{DEFINER|INVOKER} 指明谁有权限来执行存储过程，DEFINER 表示只有

定义者自己才能够执行存储过程，INVOKER 表示调用者可以执存储过程。COMMENT ‘string’是注释信息。当无任何选项说明时默认为 CONTAINS SQL。

执行存储过程的一般语法格式为：

```
CALL 存储过程名([参数[,…]])
```

其中参数的类型和个数应当与被调用的存储过程的参数定义一一对应。

1. 创建和调用不带参数的存储过程

【例 5-21】在 college 数据库中新建一个名为 p_em 的存储过程。该存储过程输出 teacher 表中所有电子系男老师的记录。

```
USE college;
DELIMITER //
CREATE PROCEDURE p_em ( )
BEGIN
    SELECT  *
    FROM teacher
    WHERE  sex = '男' and Dep='电子';
END; //
DELIMITER;
```

在 MySQL 的命令行中，默认的 SQL 语句结束符是分号“;”。当编辑完一行 SQL 语句回车后，如果语句中没有分号，则 MySQL 会认为该 SQL 语句还没编辑完成，让用户继续换行编辑；但如果语句中有分号，则 MySQL 就理解为 SQL 语句编辑完成并立即执行该语句。所以如果存储过程的程序体中包含以“;”结束的语句，那么就必须要使用 DELIMITER 语句来改变语句结束符。本例开头的“DELIMITER //”语句的作用就是把语句结束符改为“//”，当存储过程语句编辑执行完成后，再用“DELIMITER;”语句把结束符改回“;”。

【例 5-22】执行例 5-21 创建的存储过程 p_em。

```
CALL p_em ();
```

执行结果如图 5-10 所示。

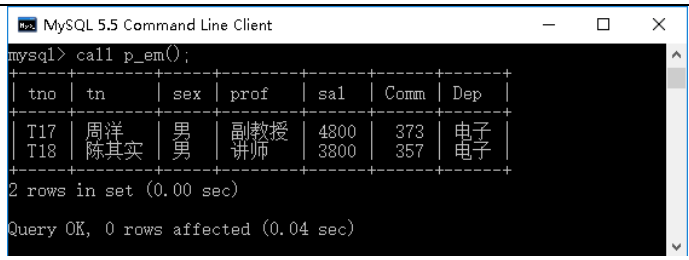

图 5-10 执行 p_em 存储过程

2. 创建和调用带有参数的存储过程

例 5-21 中创建的存储过程 p_em 只能输出电子系的男老师。如果要让用户能够按照一个任意给定的系别进行查询，这时就要用到输入参数了。

输入参数是指由调用程序向存储过程传递的参数，在创建存储过程时定义输入参数，在调用存储过程时提供相应的参数值。定义时输入参数前面加 IN 关键字

【例 5-23】在 college 数据库中创建一个名为 p_vardep 的存储过程，它能根据给定的系名查询并返回 teacher 表中相应系老师的记录。

```
USE college;
DELIMITER //
CREATE PROCEDURE p_vardep (IN depname varchar(30) )
BEGIN
    SELECT   *
    FROM teacher
    WHERE Dep=depname;
END;
//
DELIMITER;
```

其中 depname 就是存储过程 p_vardep 的一个输入参数，利用它接收调用者指定需要查询的系别名称，代入到存储过程体中的 SELECT 查询语句的 WHERE 语句中，形成为一种可变的选择条件。

在执行带输入参数的存储过程时，就必须提供相应的输入参数。传递的参数可以是值，也可以是变量。

【例 5-24】用传值的方式执行存储过程 p_vardep，查找物联网系的老师。

```
call p_vardep('物联网');
```

运行结果如图 5-11 所示。

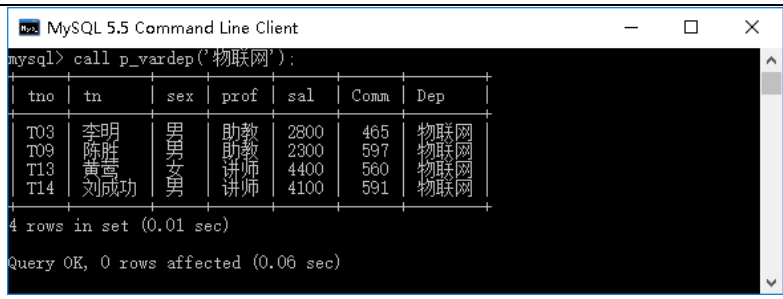

图 5-11 传值调用带输入参数的存储过程

【例 5-25】用传变量的方式执行存储过程 p_vardep，查询信息管理系的老师。

```
SET @seekdep='电子';
CALL p_vardep(@seekdep);
```

首先用一条 SET 语句定义了一个会话变量@seekdep，值为“电子”，然后将其作为输入参数传递给被调用的存储过程 p_vardep。运行结果如图 5-12 所示。

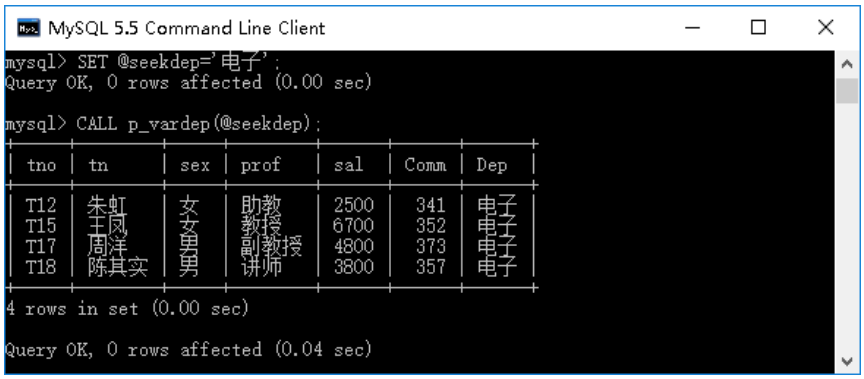

图 5-12 传变量调用带输入参数的存储过程

与输入参数相反，如果需要从存储过程中返回数据给调用者，则可以使用输出参数。输出参数在定义时前面用 OUT 关键字表示。

【例 5-26】在 college 数据库中创建存储过程 p_exist，其功能是判断指定姓名的老师是否存在。如果存在则返回“Yes”，否则返回“No”。

```
USE college;
DELIMITER //
CREATE PROCEDURE p_exist ( IN tname VARCHAR ( 20 ), OUT rt VARCHAR ( 3 ) )
BEGIN
DECLARE cnt INT;
SELECT count( * ) INTO cnt FROM teacher WHERE tn = tname;
```

```
    IF cnt>0 THEN
        set rt='Yes';
    ELSE
        set rt='No';
    END IF;
    END
    //
    DELIMITER ;
```

存储过程中首先用一条 DECLARE 语句定义了一个变量 cnt，然后使用带 INTO 短语的 SELECT 语句将姓名等于输入参数 tname 的值的老师的人数存入 cnt，接下来根据 cnt 的值设置输出参数 rt 的值。

【例 5-27】执行存储过程 p_exist，根据传出参数查询是否存在姓名为“TOM”的老师和姓名为“李民”的老师。

```
    set @r = NULL;
    call p_exist('TOM',@r);
    mysql> select @r;
    mysql> call p_exist('李民',@r);
    mysql> select @r;
```

在调用存储过程之前，需要设置一个变量用于接收存储过程的输出参数。调用存储过程完成后，再使用该变量显示返回的结果。

运行结果如图 5-13 所示。通过会话变量@r 接收 p_exist 执行后传出的参数，从而可知传入“TOM”后传出的结果为“No”，说明没有姓名为“TOM”的老师；而传入“李民”后传出的结果为“Yes”，说明存在名为“李民”的老师。

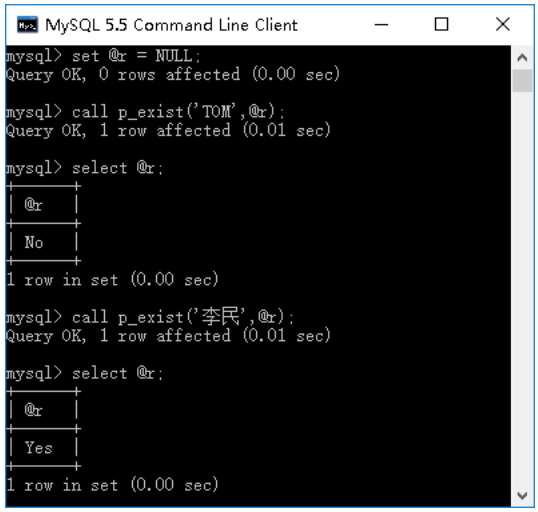

图 5-13 调用带输出参数的存储过程

5.3.3 查看存储过程

在 MySQL 中，有 4 种方法可以查看存储过程的名称、所属数据库、定义、创建时间等信息。

(1) 使用“SHOW PROCEDURE STATUS [WHERE 条件]”命令可以查看系统内的所有或者部分满足条件的存储过程的名称、所在数据库、创建时间、作者等信息。

【例 5-28】查询 college 数据库里有哪些存储过程。

```
SHOW PROCEDURE STATUS  WHERE Db='college';
```

运行结果如图 5-14 所示。

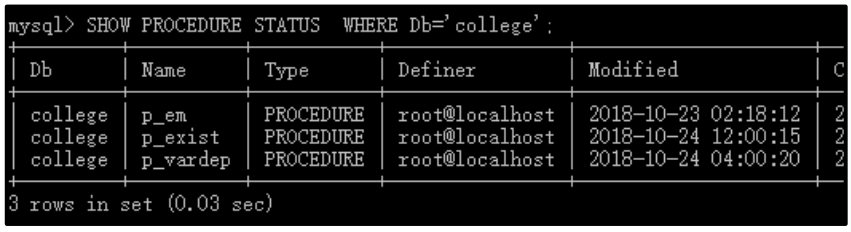

图 5-14 SHOW PROCEDURE STATUS 命令示例

(2) 使用“SHOW CREATE PROCEDURE 数据库.存储过程名;”命令可以查看指定存储过程的定义语句等信息。

【例 5-29】显示 college 数据库的 p_em 存储过程的定义。

```
SHOW CREATE PROCEDURE college.p_em \G;
```

命令最后的\G 参数用于使结果纵向显示，便于阅读。运行结果如图 5-15 所示。

图 5-15 SHOW CREATE PROCEDURE 命令示例

(3) 存储过程创建后被存储在 information_schema 数据库的 ROUTINES 表中，所以可以使用 SELECT 命令来查询 information_schema.ROUTINES 表获得存储过程的信息。

【例 5-30】用查询 information_schema.ROUTINES 方式显示所有存储过程的名字、类型和所属数据库。

```
SELECT ROUTINE_NAME,ROUTINE_TYPE,ROUNTINE_SCHEMA
FROM information_schema.ROUTINES ;
```

运行结果如图 5-16 所示。

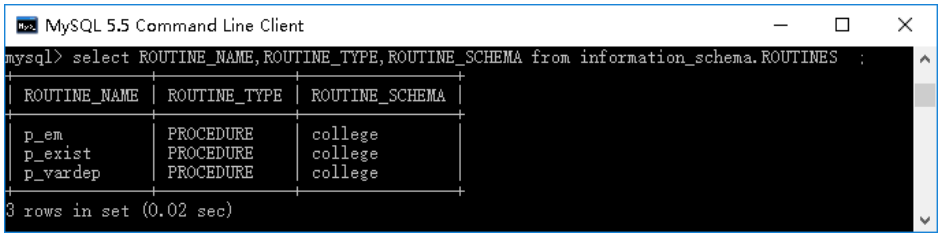

图 5-16 查询 information_schema.ROUTINES 示例

(4) 创建好的存储过程的源代码被存放在系统数据库 mysql 的 proc 表中，所以可以可以使用 SELECT 命令来查询 mysql.proc 表获得存储过程的信息。

【例 5-31】利用 mysql.proc 表查询 p_em 存储过程的详细信息。

```
SELECT ROUTINE_NAME,ROUTINE_TYPE,ROUNTINE_SCHEMA
FROM information_schema.ROUTINES ;
```

运行结果如图 5-17 所示。

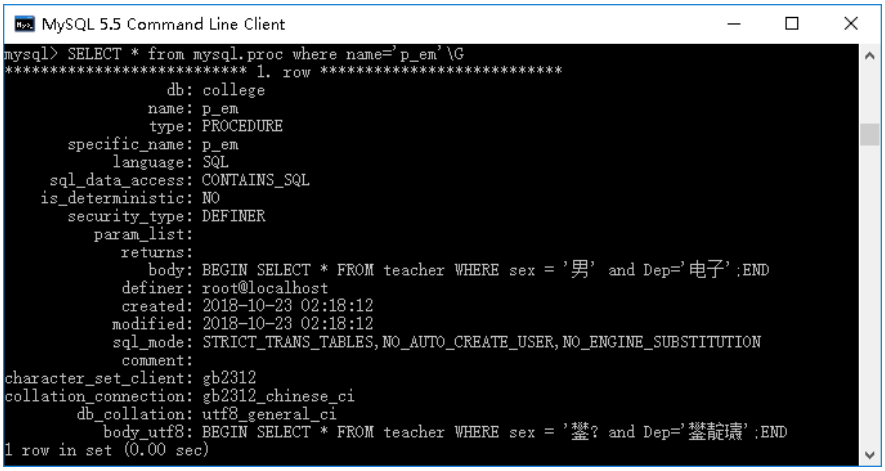

图 5-17 查询 mysql.proc 示例

5.3.4 删除存储过程

在 MySQL 中删除存储过程的语句的语法格式为：

```
DROP PROCEDURE [IF EXISTS] 存储过程名;
```

其中可选项 IF EXISTS 短语的作用是在删除时如果被删除的存储过程并不存在，系统也不用抛出因找不到存储过程而删除失败的错误。

【例 5-32】删除 college 数据库中的存储过程 p_em。

```
USE college;
DROP PROCEDURE p_em;
```

另外，在 MySQL 中有一条用于修改存储过程的语句 ALTER PROCEDURE，但该语句只能修改存储过程的读写权限等特性，而不能修改存储过程的程序体。所以如果要修改存储过程，主要是通过删除存储过程再重新定义的方式来解决。

5.4 触发器

5.4.1 触发器概述

触发器是数据库中一种特殊的存储过程。触发器与存储过程的区别在于触发器能够自动执行并且不含有参数。每一个触发器都定义有触发事件和触发程序。每当数据库管理系统监测到这些触发事件发生时，就会自动去执行相应的触发程序。

理论上数据库中所有的对象都可以设置触发器。但不同的数据库管理系统对触发器的支持各不相同。比如 SQL SERVER 2008 以上的版本中就有用于数据定义（CREATE、

ALTER、DROP）时触发的 DDL 触发器、数据更新（INSERT、UPDATE、DELETE）时触发的 DML 触发器和用于系统安全管理的登录触发器等。而 MySQL 是在 5.0 版本才开始出现触发器，而且仅支持 DML 触发器。

最常见的触发器就是 DML 触发器。每一个数据表都可以设置 DML 触发器。其触发事件就是对表数据的更新操作。当对该表进行 INSERT、UPDATE、DETETE 操作时，将激活触发器。如无特别的说明，本书后面内容中的触发器就是 DML 触发器。

触发器的优点主要表现在：

(1) 只要触发事件发生，相应操作的触发器就会立即自动执行，从而不会出现当事件发生后该做的操作被忘记或忽略的情况，提高了数据管理的效率和完整性。

(2) 触发器的执行单元可以实现任意的数据操作，可以引用其它表中的字段，这些能力使得触发器可以实现更为复杂的数据完整性要求。

(3) 触发器激活后可以获取到表数据修改前后的状态，因而可以根据其差异采取相应的措施。

(4) 触发器可以用程序方式检验和判断数据操作的结果，从而可以防止恶意的或错误的 INSERT、UPDATE 和 DELETE 操作。

5.4.2 创建触发器

在 MySQL 中创建触发器的语句的语法格式为：

```
CREATE TRIGGER 触发器名
AFTER/BEFORE INSERT/UPDATE/DELETE ON 表名
FOR EACH ROW
BEGIN
   语句序列;
END;
```

说明：

(1) INSERT/UPDATE/DELETE 为三种触发事件：

- INSERT：在表中插入新记录时。
- UPDATE：更改表中的记录时。
- DELETE：从表中删除记录时。

(2) BEFORE/AFTER 为触发时间。BEFORE 表示在触发事件发生后先去执行触发程序，然后再去执行数据更新操作，适用于在实际的数据更新之前事先做一些检查和处

理的情况；AFTER 表示在引起触发事件的操作成功执行后才去执行触发程序，适用于更新完成后自动再完成其它一些操作的情况。

(3) FOR EACH ROW 表示行级触发器，即数据更新操作影响的每一条记录都会执行一次触发程序。

(4) 在触发程序中有两个特殊的系统对象 OLD 和 NEW，用于存放被更新的记录，供触发程序的语句引用。

- OLD 和和 NEW 仅能在触发程序中使用，其它地方使用无效。
- OLD 表示被删除的记录。可以使用“OLD.字段名”访问被删除记录中的某个字段。OLD 是只读的，只能引用它，不能更改它。
- NEW 表示新插入的记录。可以使用“NEW.字段名”访问新记录中的某个字段。在 BEFORE 触发程序中，由于是在实际的插入操作之前，所以可使用 SET 命令更改 NEW 记录的值。但在 AFTER 触发程序中，因为已经处于实际的插入操作之后，因此不能使用再修改 NEW 记录的值。
- 对于 INSERT 操作，只有 NEW 关键字是合法的。对于 DELETE 操作，只有 OLD 关键字是合法的。对于 UPDATE 操作，实际上删除旧记录而增加修改后的新记录，所以 NEW 关键字和 OLD 关键字都可用。

【例 5-33】为 college 数据库中的 teacher 表创建一个名为 tr_teacher_insert 的触发器，实现不允许给 teacher 表新插入工资为负值的记录。

参考的触发器设计如下：

```
USE college;
DELIMITER //
CREATE TRIGGER tr_teacher_insert BEFORE INSERT ON teacher
FOR EACH ROW
BEGIN
    IF NEW.sal is NOT NULL && NEW.sal<0 THEN
        SIGNAL SQLSTATE 'HY000'
        SET MESSAGE_TEXT = '工资不能小于 0，插入失败！';
    END IF;
END;
//
DELIMITER;
```

根据需求可知在实际的插入操作进行之前就应该进入触发程序判断情况，所以触发时间设计为 BEFORE，触发事件为 INSERT。进入触发程序后，新增的记录被系统保存在 NEW 对象中，根据 NEW.sal 就可得到新增老师的工资，如果它不为空并且小于 0，则说明不符合约束，执行 SINGNAL SQLSTATE 命令抛出异常信息，中止了后续操作，从而达到了防止工资为负的记录被插入到表中的目的。而 NEW.sal 为空或者不小于 0，则触发程序没有其它操作，去执行原计划的实际插入操作。

在 tr_teacher_insert 触发器创建好以后，使用两组 INSERT 语句插入新记录并使用 SELECT 语句对插入情况进行检查，如图 5-18 所示。从结果可见工资为负的记录插入失败，而非负工资的记录被正常插入。

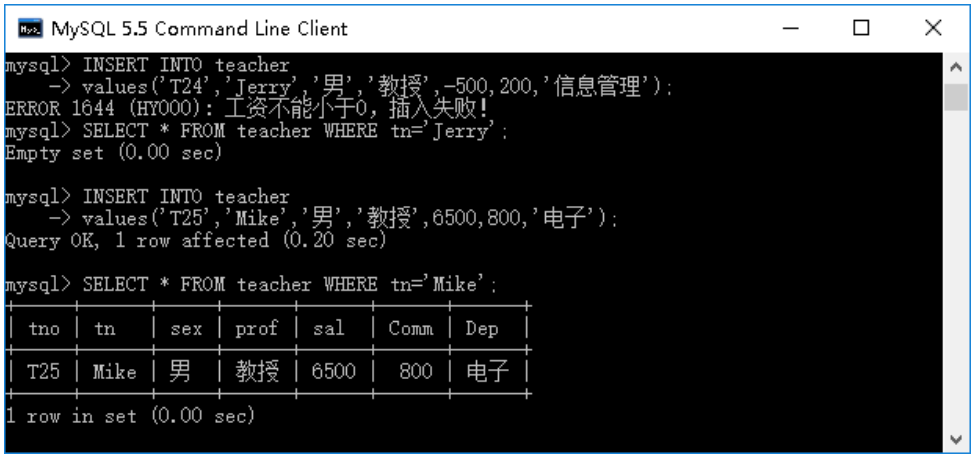

图 5-18 测试 tr_teacher_insert 触发器

【例 5-34】为 college 数据库中的 teacher 表创建一个名为 tr_teacher_delete 的触发器，实现删除 teacher 表的记录后，自动将这些老师在 tc 表的授课记录一并删除。

参考的触发器设计如下：

```
USE college;
DELIMITER //
CREATE TRIGGER tr_teacher_delete AFTER DELETE ON teacher
FOR EACH ROW
BEGIN
    DELETE FROM tc WHERE tc.tno=OLD.tno;
END;
//
DELIMITER;
```

根据需求可知应该在 teacher 表中实际的删除操作完成后再进入触发程序，所以触发时间设计成了 AFTER，触发事件为 DELETE。进入触发程序后，被删除的记录被保存在 OLD 对象中，根据 OLD.tno 就可得到被删除老师的工号，利用它构建删除 tc 表的 DELETE 命令的条件，从而删除了这些老师的授课记录。

如图 5-19 所示，在成功创建 tr_teacher_delete 触发器以后，首先用 SELECT 语句查询到工号为 T25 的老师有两门课程，在使用 DELETE 语句将该老师删除后，再次用 SELECT 语句查询 tc 表则可见相应的授课记录也被删除了。

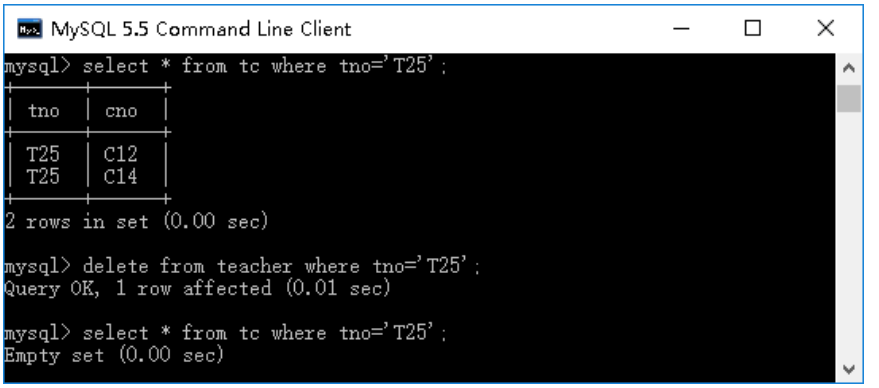

图 5-19 测试 tr_teacher_delete 触发器

【例 5-35】为 college 数据库中的 teacher 表创建一个名为 tr_teacher_update 的触发器，确保在修改老师的工资时，老师的工资只能增加，且工资增加多少，补助就增加多少。为了简单起见，不考虑工资为空的情况。

参考的触发器设计如下：

```
USE college;
DELIMITER //
CREATE TRIGGER tr_teacher_update BEFORE UPDATE ON teacher
FOR EACH ROW
BEGIN
    IF NEW.sal < OLD.sal THEN
        SIGNAL SQLSTATE 'HY000'
        SET MESSAGE_TEXT = '新工资不能小于原工资，修改失败！';
    END IF;
    SET NEW.Comm = NEW.Comm + (NEW.sal-OLD.sal);
END;
```

```
//
DELIMITER;
```

根据需求可知应该在 teacher 表中实际的修改操作前进入触发程序，所以触发时间设计为 BEFORE，触发事件为 UPDATE。进入触发程序后，修改前的记录保存在 OLD 中，修改后的保存在 NEW 中。如果新工资 NEW.sal 小于了旧工资 OLD.sal，则说明违反了约束，执行 SINGNAL SQLSTATE 命令抛出异常信息，中止后续操作，修改以失败告终；如果 NEW.sal 不于小 OLD.sal，则使用 SET 命令设置好新的补助值 NEW.Comm，然后结束触发程序去执行实际的修改操作，即利用 OLD 删除表中的旧记录，将最终的 NEW 记录插入到表中。

在 tr_teacher_update 触发器创建好以后，使用两组 UPDATE 语句修改记录并使用 SELECT 语句对修改情况进行检查，如图 5-20 所示。从结果可见工资从 3800 改为 3000 失败，而改为 3900 就不仅成功修改了工资，同时也增加了补助。

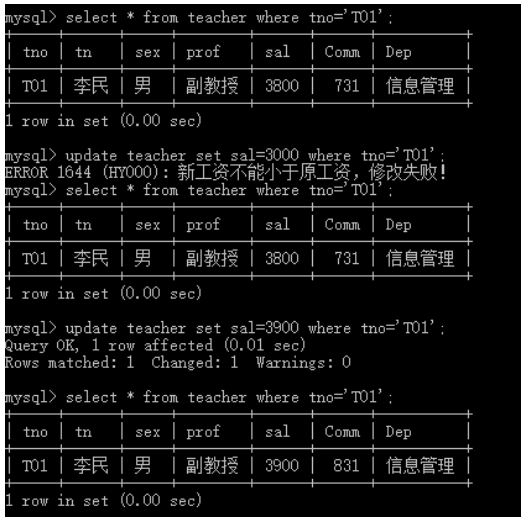

图 5-20 测试 tr_teacher_update 触发器

5.4.4 查看触发器

在 MySQL 中一般使用三种方式查看触发器的定义。

(1) 使用 SHOW TRIGGERS 命令查看某些触发器的详细信息。

语法格式为：

```
SHOW TRIGGERS
[FROM 数据库名 | IN 数据库名] [LIKE 表名模式 | WHERE 条件];
```

说明：

- 可以使用 FROM/IN 短语指定查询某个数据库中的触发器。缺省为当前数据库。
- 默认查询数据库中的全部触发器。
- 如果只想查看定义在某些表上的触发器，可以使用 LIKE 短语。但要注意这里的模式只与表名匹配而不是与触发器的名字匹配。
- 如果只想查询满足条件的一部分触发器，可以使用 WHERE 短语。

【例 5-36】查看 college 数据库中的所有触发器定义。

```
SHOW TRIGGERS FROM college;
```

【例 5-37】查看当前数据库中所有触发事件为 UPDATE 的触发器。

```
SHOW TRIGGERS WHERE Event='UPDATE';
```

【例 5-38】查看 college 数据库中 teacher 表上定义的所有触发器。

```
SHOW TRIGGERS IN college LIKE '%teacher%';
```

(2) 使用 SHOW CREATE TRIGGER 命令查看某个触发器的定义。

语法格式：

```
SHOW CREATE TRIGGER 触发器名;
```

【例 5-39】查看 college 数据库中名为 tr_teacher_insert 触发器的定义。

```
SHOW CREATE TRIGGERS tr_teacher_insert;
```

5.4.5 删除触发器

删除触发器的语句的语法格式为：

```
DROP TRIGGER [IF EXISTS] 触发器名;
```

其中可选项 IF EXISTS 短语的作用是如果不存在被删除的触发器系统也不用抛出因找不到触发器而删除失败的错误。

【例 5-40】删除 college 数据库中 teacher 表上的 tr_teacher_insert 触发器。

```
USE college;
DROP TRIGGER tr_teacher_insert;
```

5.5 游标*

数据库中使用 SQL 语句对数据的操作是一次一集合的方式。但如果我们希望对 SELECT 命令的结果集按一次一记录的方式逐条处理，这时就可以使用游标来实现。

游标就是将 SELECT 语句查询结果转换成了一个记录序列，并定义了一个指向一条记录的指针来指示游标中的当前记录位置。每次就可以借助游标指针提取当前记录的

数据，实现对一行记录的查询或处理。通过移动指针就可以变换当前要处理的记录。

使用游标必须经历声明游标、打开游标、提取记录（可通过变换游标指针多次提取）和关闭游标的程序化过程，一般不能单独使用，常常是在存储过程或触发器中出现。

5.5.1 游标的使用

1.声明游标

要使用游标对查询结果集中的数据进行处理，首先需要声明游标。MySQL 声明游标的语法格式为：

```
DECLARE 游标名 CURSOR FOR SELECT 语句;
```

例如要声明一个名为 cur_teacher 的游标，其中包含电子系教师的工号(tno)、姓名(tn)和性别(sex)记录，相应的语句如下：

```
DECLARE cur_teacher CURSOR
FOR SELECT tno,tn,sex FROM teacher  WHERE  Dep='电子';
```

2.打开游标

游标必须要先打开再使用。所谓打开游标，就是按游标声明中的 SELECT 语句准备数据，初始化记录指针。

在 MySQL 中使用 OPEN 命令打开游标，其语法格式为：

```
OPEN 游标名 ;
```

例如打开名为 cur_teacher 的游标的命令如下：

```
OPEN cur_teacher;
```

3.提取游标数据

提取游标数据是指从游标中取出当前记录。MySQL 提取游标数据使用 FETCH 命令，其语法格式为：

```
FETCH 游标名 INTO 变量 1{,变量 2…}
```

其中，变量的个数必须与游标声明中 SELECT 语句查询结果的字段数一致，一一对应地存储从游标当前记录中提取出的字段值。

参考前面声明的游标 cur_teacher，假设已经事先定义好了相应的变量，则可以用下面的语句来提取游标当前记录中的教师工号、姓名和性别分别存入变量 ct_no、ct_name 和 ct_sex 中：

```
FETCH cur_teacher INTO ct_no, ct_name, ct_sex;
```

每执行一次 FETCH 语句从查询结果集中取出一条记录，将该记录字段的值送入指

定的变量，并且记录指针会自动移到下一条记录。通过循环结构就可以逐条访问查询结果集中的所有记录。

4.游标的关闭

游标使用完毕后必须关闭它。MySQL 中关闭游标的语句的语法格式为:

```
CLOSE  游标名;
```

例如，关闭 cur_teacher 游标的命令为：

```
CLOSE cur_teacher ;
```

游标关闭之后就不能再使用 FETCH 语句从游标查询结果集中取出数据了。

5.5.2 游标应用示例

首先用 SELECT 命令查询出电子系老师的信息，如图 5-21 所示：

```
mysql> SELECT * FROM teacher WHERE dep='电子';
+-----+--------+-----+--------+------+------+------+
| tno | tn     | sex | prof   | sal  | Comm | Dep  |
+-----+--------+-----+--------+------+------+------+
| T12 | 朱虹   | 女  | 助教   | 2500 |  341 | 电子 |
| T15 | 王凤   | 女  | 教授   | 6700 |  352 | 电子 |
| T17 | 周洋   | 男  | 副教授 | 4800 |  373 | 电子 |
| T18 | 陈其实 | 男  | 讲师   | 3800 |  357 | 电子 |
+-----+--------+-----+--------+------+------+------+
4 rows in set (0.01 sec)
```

图 5-21 电子系老师数据示例

现在要查询出奇数行老师的工资的总和，使用 SQL 的 SELECT 命令直接实现就很困难了。此时就可以考虑创建一个使用游标的存储过程来解决。相应的代码如下：

```
USE college;
delimiter //
create procedure p_odd()
begin
    DECLARE isover INT DEFAULT 0;    -- 用于判断是否还能正确读取
    DECLARE n INT DEFAULT 0;   -- 用于计数
    DECLARE allSal INT DEFAULT 0;    -- 存放总工资
    DECLARE ct_sal INT DEFAULT 0;    -- 用于存放读出的工资


    DECLARE cur_teacher CURSOR FOR       -- 声明游标
    SELECT sal FROM teacher
     WHERE dep='电子';
```

```
    -- 捕获没记录可读错误，用于配合后面的循环
    DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET isover = 1;

    OPEN cur_teacher;   -- 打开游标
    FETCH cur_teacher INTO ct_sal; -- 提取首条记录
    WHILE isover != 1 DO
        SET n = n+1;  -- 计数器加 1
        IF n%2=1 THEN   -- 是奇数行记录
            SET allSal = allSal+ct_sal;
        END IF;
        FETCH cur_teacher INTO ct_sal; -- 提取下一条记录
    END WHILE;
    SELECT allSal;     -- 查询最终的结果
    CLOSE cur_teacher; -- 用 CLOSE 语句把光标关闭”
end;
//
delimiter ;
```

存储过程在执行时首先定义需要的变量和游标，然后从打开的游标中逐条读取记录，利用变量 n 存储当前记录的行号。根据 n 整除 2 的余数可以判断出读取的是否是奇数行。是奇数行就累加总工资 allSal。程序中只要 FETCH 命令无记录可读时就会被 CONTINUE HANDLE 捕获，将 isover 设置为 1。WHILE 语句就可以根据 isover 判断是否继续读取记录。程序的运行结果如图 5-22 所示。

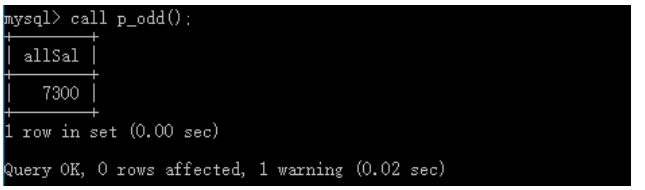

图 5-22 统计奇数行电子系老师工资的结果示例

这个结果表明统计了第 1 行老师的工资 2800 和第 3 行老师的工资 3500。

5.6 小结

本章主要讲述了在数据库系统中的除数据表以外的其他数据库对象，包括索引、视图、存储过程和触发器等。索引是一种特殊类型的数据对象，它可以用来提高对表中数据的访问速度而且还能够对表实施完整性约束。索引类型一般包括普通索引、唯一索引全文索引等。视图是从一个或多个表中导出来的表，是一种虚表。视图的结构和数据依赖于基本表。视图可以简化 SQL 查询语句的编写，提高数据库的安全性。存储过程可以将 SQL 语句和控制流语句预编译到集合并保存到服务器端，它使得管理和操作数据库更为方便。触发器是一种特殊类型的存储过程，在某个指定的事件发生时被激活，常常用于扩展数据的完整性。游标就是将 SELECT 语句查询结果转换成了一个记录序列，并定义了一个指向一条记录的指针来指示游标中的当前记录位置。每次就可以借助游标指针提取当前记录的数据，实现对一行记录的查询或处理。游标常用于解决在数据库操作时需要一次一记录的操作方式的问题。

5.7 习题

1. 如何理解索引？索引越多越好吗？
2. 索引关键词的选取原则有哪些？
3. 索引分为哪些种类？什么是全文索引？
4. 索引与约束有什么关系？
5. 视图与基表有什么区别和联系？视图与 SELECT 语句有什么关系？
6. 编写一个存储过程实现提供的学生的姓名参数查询他学的课程名和成绩。
7. MySQL 触发器中的触发事件有几种？
8. MySQL 触发器中的触发时间有几种？
9. 编写一个触发器实现禁止删除自动化系的老师。