# 16 视图应该怎样去应用和管理呢？

更新时间：2020-04-08 15:55:47

> 成功的奥秘在于目标的坚定。——迪斯雷利

MySQL 视图是个非常特殊的存在，之所以说它特殊，是因为它非常好用、非常有价值，但是常常被忽略。这个结论的得出来自于面试经验，我很惊讶的发现，很多工作了两三年的同学，竟然没有用过视图（对视图的概念、使用方法知之甚少），这实在是不应该了。这一节里，我将会对视图做详细的解读，包括视图的含义、作用、使用方法等等。

## 1. 初识视图

视图与数据表有着类似的属性和结构，所以，我们可以像操作表（CRUD）那样来操作视图。那么，在讲解视图的具体操作方法之前，我们先来搞懂什么是视图、它有哪些作用、以及相对于数据表来说，它有会有哪些优势。

### 1.1 视图的定义

视图是从数据库中一个或多个表（注意，这是重点）中导出来的表，但是，它是一个虚拟表。另外，视图还可以在已经存在的视图之上再去定义。也就是说，视图的来源可以是表，也可以是视图。

视图与表一样，只要定义之后便会存储在数据库中，但是它与表最大的不同是：视图并不存储数据，我们通过视图看到的数据只是存放在表中的数据。正是由于视图的基本结构与属性与数据表是类似的，所以，我们可以对它进行查询、修改、删除等操作。最后，还需要知道视图与表之间的联系：

- 当对视图中的数据进行修改，对应表中的数据也会被修改
- 当对表中的数据进行修改，与它相关联的视图数据也会发生变化

可以看到，视图的定义并不复杂，但是这里面所涉及的内容确是不少。好好理解关于视图的定义，特别是它与表之间的联系，这将在接下来讲解视图操作时起到非常重要的指导作用。

#### 1.2 视图的作用

其实，关于视图有什么用，真的不好解释。下面，我们先去创建两张表（注意，这是数据表，不是视图），之后，再通过表去把视图的作用说清楚。

```
CREATE TABLE `imooc_mysql`.`fruit_order` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT 'id',
  `count` int(11) NOT NULL COMMENT '个数',
  `fruit_id` bigint(20) NOT NULL COMMENT '关联 fruit 表 id',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='水果订单表';

CREATE TABLE `imooc_mysql`.`fruit` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT 'id',
  `name` varchar(32) NOT NULL COMMENT '名称',
  `price` int(11) NOT NULL COMMENT '价格',
  `extra` varchar(256) COMMENT '备注',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='水果表';
```

我在 imooc_mysql 库（这并没有特别的含义）中创建了两张表：fruit_order 和 fruit，需要特别注意的是表中的各个字段有的是 NOT NULL，而有的并没有标记。我建议不要随意更改这两张表的定义，因为后面的内容讲解都会使用到它们。目前，我有两个需求：

- 对于 fruit，我不想让运营人员看到备注（extra）信息
- 对于 fruit_order，我想要看到水果的名称（fruit.name）和总价格（fruit_order.count * fruit.price）

虽然，我可以通过连接多表、权限设置的方式实现这两个需求，但是，操作过程会非常繁琐。而视图则提供了一个很好的解决办法，创建视图可以取自表的部分信息、多表数据之间的计算等等，这样既能简单的满足需求，也不会破坏原表的结构。

#### 1.3 视图的优势

视图最直接的优势就是简单，它不仅可以简化用户对数据的理解，也可以最大程度的简化用户的操作（想想复杂的连接，那可真是噩梦）。除了简单之外，视图还最大程度的保证了数据的安全性，这主要体现在以下四点：

- 使用权可被限制在母表（基于这张表创建的视图）的行的子集上
- 使用权可被限制在母表的列的子集上
- 使用权可被同时限制在母表的行和列的子集上
- 使用权可被限制在多个母表的连接所限定的行上

理解了视图的概念、思想，以及知道了它能用来做什么，接下来，我们就一起去看一看怎样操作、使用视图，并从中发现视图更多的特性。

## 2. 创建视图

视图中的数据来自于真实的数据表，所以，可以猜想创建视图需要使用到 SELECT 语句去查询表。这里，我先去解读创建视图的语法，之后，再去创建视图完成之前的两个需求。

#### 2.1 创建视图的语法解读

在谈到视图定义的时候就说过，视图可以基于一张表创建，也可以基于多张表创建，那么，在 MySQL 中到底应该怎样创建视图呢 ？我们先来看一看视图的创建语法：

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
  VIEW view_name [(column_list)]
  AS SELECT_statemment
  [WITH [CASCADED | LOCAL] CHECK OPTION]
```

官网给出的这个语法确实晦涩难懂，下面，我将对其中的关键字进行解读：

- CREATE 表示创建新的视图、REPLACE 表示替换已经创建的视图
- ALGORITHM 表示视图选择的算法，取值有三个
    - UNDEFINED：让 MySQL 来自动选择算法
    - MERGE：将使用的视图语句和定义合并起来，使视图定义的某一部分取代语句对应的部分
    - TEMPTABLE：将视图的结果存入临时表，之后使用临时表来执行语句
- view_name 指定视图的名称、column_list 指定视图的属性列
- SELECT_statemment 表示 SELECT 语句，也就是视图的母表查询语句
- WITH CHECK OPTION 表示视图在更新时保证在视图的权限范围内，其中：
    - CASCADED：默认值，表示更新视图时要满足所有相关视图和表的条件
    - LOCAL：表示更新视图时满足该视图本身定义的条件即可

另外，需要注意，创建视图需要有 CREATE VIEW 权限，以及针对母表 SELECT 的权限。如果是 CREATE OR REPLACE 还要求对视图有 DROP 的权限。

#### 2.2 创建视图实操

我们在创建视图之前，先去做一些准备工作，即在 fruit_order 和 fruit 表中插入一些数据：

```
-- 向 fruit 表中插入数据
INSERT INTO `fruit`(`name`, `price`, `extra`) VALUES('苹果', 5, 'a');
INSERT INTO `fruit`(`name`, `price`, `extra`) VALUES('香蕉', 6, 'b');
INSERT INTO `fruit`(`name`, `price`, `extra`) VALUES('大枣', 7, 'c');
INSERT INTO `fruit`(`name`, `price`, `extra`) VALUES('葡萄', 8, 'd');

-- 向 fruit_order 表中插入数据， 注意，fruit_id 要与 fruit 相对应
INSERT INTO `fruit_order`(`count`, `fruit_id`) VALUES(12, 1);
INSERT INTO `fruit_order`(`count`, `fruit_id`) VALUES(8, 2);
INSERT INTO `fruit_order`(`count`, `fruit_id`) VALUES(7, 3);
INSERT INTO `fruit_order`(`count`, `fruit_id`) VALUES(15, 4);
```

fake 数据之后，我们先来处理第一个需求：对于 fruit，我不想让运营人员看到备注（extra）信息。创建如下视图：

```
mysql> CREATE VIEW `imooc_mysql`.`fruit_v` AS SELECT name, price FROM `imooc_mysql`.`fruit`;
Query OK, 0 rows affected (0.03 sec)
```

以上语句创建了视图 fruit_v，后缀 v 标识这是视图，但这只是我的个人习惯，可以忽略。fruit_v 的母表是 fruit，且限制使用 fruit 中的两个列。创建完成之后，我们可以去看一看 fruit_v 中的数据：

```
mysql> SELECT * FROM fruit_v;
+--------+-------+
| name   | price |
+--------+-------+
| 苹果   |     5 |
| 香蕉   |     6 |
| 大枣   |     7 |
| 葡萄   |     8 |
+--------+-------+
4 rows in set (0.00 sec)
```

正如之前所说，视图中的数据是来自于数据表的。在默认情况下（未显示的指定），视图的字段和母表的字段是一致的，当然你也可以自己来指定。同时，你也可以看到，对于这个需求，使用视图来处理是多么的简单方便。

对于第二个需求，需要使用到两张表的数据，且包含了一个数据计算。可以使用以下语句来创建视图并验证结果（创建语句太长，做了格式化处理）：

```
-- 创建视图 fruit_order_v，SELECT 查询两张表
mysql> CREATE VIEW `imooc_mysql`.`fruit_order_v` (
    ->     name ,
    ->     count ,
    ->     price ,
    ->     total
    ->   ) AS SELECT
    ->     name,
    ->     count,
    ->     price,
    ->     count*price
    ->   FROM
    ->     fruit,
    ->     fruit_order
    ->   WHERE
    ->     fruit_id=fruit.id;
Query OK, 0 rows affected (0.04 sec)

-- 查看 fruit_order_v，验证是否满足需求
mysql> SELECT * FROM fruit_order_v;
+--------+-------+-------+-------+
| name   | count | price | total |
+--------+-------+-------+-------+
| 苹果   |    12 |     5 |    60 |
| 香蕉   |     8 |     6 |    48 |
| 大枣   |     7 |     7 |    49 |
| 葡萄   |    15 |     8 |   120 |
+--------+-------+-------+-------+
4 rows in set (0.00 sec)
```

经过对这两个例子的介绍，相信你对视图的理解应该是更进一步了。以此为基础，想一想你在工作中遇到的不好处理的数据，是否可以使用视图来简化操作，提升工作效率。

## 3. 查看视图

对于你创建或参与创建的视图，你当然会知道它的字段属性、结构信息等等。但是，对于之前已经存在的或其他人创建的视图，我们想要查看它的相关信息应该怎么办呢 ？这里，我将会说明两种方式查看视图的定义。

### 3.1 使用 SHOW 语句查看视图

如果想看视图的列属性定义，可以直接使用 DESC 语句。但是，结果中只包含字段定义、数据类型、是否允许为 NULL，默认值等基础信息。如果想要查看视图更加详细的信息，就需要使用 SHOW 语句。例如：

```
-- SHOW TABLE STATUS 语法
SHOW TABLE STATUS LIKE 'view name';

-- 查看 fruit_v 的信息
mysql> SHOW TABLE STATUS LIKE 'fruit_v'\G
*************************** 1. row ***************************
           Name: fruit_v
         Engine: NULL
        Version: NULL
     Row_format: NULL
           Rows: NULL
 Avg_row_length: NULL
    Data_length: NULL
Max_data_length: NULL
   Index_length: NULL
      Data_free: NULL
 Auto_increment: NULL
    Create_time: NULL
    Update_time: NULL
     Check_time: NULL
      Collation: NULL
       Checksum: NULL
 Create_options: NULL
        Comment: VIEW
1 row in set (0.00 sec)
```

可以看到，除了 Name 和 Comment 之外，其他的都是 NULL，这是因为视图是一个虚表。我们通过查询这条 SHOW 语句的主要目的是确定当前操作的是视图（Comment 字段）而不是数据表（可以使用这条语句查询下数据表，对比下视图的数据返回）。

如果你想知道当前的视图是怎么定义出来的，可以这样：

```
-- SHOW CREATE VIEW ViewName 用于查看视图定义，注意，视图名不要加引号
mysql> SHOW CREATE VIEW fruit_v\G
*************************** 1. row ***************************
                View: fruit_v
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `fruit_v` AS select `fruit`.`name
` AS `name`,`fruit`.`price` AS `price` from `fruit`
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set (0.00 sec)
```

可以看到，`SHOW CREATE VIEW` 语句打印了视图名、视图定义（包含默认选项）以及字符集信息。

### 3.2 在 information_schema.VIEWS 表中查看视图

我在之前的小节中说过，SHOW 语句的信息其实都来自于表数据，而视图的信息则记录在 information_schema 系统库中的 VIEWS 表中。这张表中存储了所有的视图定义，可以通过查询它来查看视图的详细信息。如下所示：

```
mysql> SELECT * FROM information_schema.VIEWS WHERE TABLE_SCHEMA = 'imooc_mysql' AND TABLE_NAME = 'fruit_v'\G
*************************** 1. row ***************************
       TABLE_CATALOG: def
        TABLE_SCHEMA: imooc_mysql
          TABLE_NAME: fruit_v
     VIEW_DEFINITION: select `imooc_mysql`.`fruit`.`name` AS `name`,`imooc_mysql`.`fruit`.`price` AS `price` from `imooc_mysql`.`fruit`
        CHECK_OPTION: NONE
        IS_UPDATABLE: YES
             DEFINER: root@localhost
       SECURITY_TYPE: DEFINER
CHARACTER_SET_CLIENT: utf8
COLLATION_CONNECTION: utf8_general_ci
1 row in set (0.01 sec)
```

# 4. 修改视图

当我们遇到已经存在的视图不满足需求了、母表的字段发生变化了等等情况，就需要去修改视图的结构。为此，MySQL 提供了两种修改方法：`CREATE OR REPLACE VIEW` 和 `ALTER`。下面，我们先去看一看它们的语法，然后再去使用它们。

### 4.1 修改视图的两种语法

首先，来看一看 `CREATE OR REPLACE VIEW` 语句的语法：

```
CREATE OR REPLACE [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
  VIEW view_name [(column_list)]
  AS SELECT_statemment
  [WITH [CASCADED | LOCAL] CHECK OPTION]
```

你可以发现，创建视图和修改视图的语句是完全一致的。它所表达的语义是：当视图已经存在，修改语句对视图实现修改（注意，需要有对应的权限）；当视图不存在时，则创建视图。

`ALTER` 语句则不同，它需要保证你提供的视图是存在的。语法如下：

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
  VIEW view_name [(column_list)]
  AS SELECT_statemment
  [WITH [CASCADED | LOCAL] CHECK OPTION]
```

你又发现了，除了第一个关键字之外，它与创建视图的语法又是一样的，所以，我这里不再赘述了。

### 4.2 修改视图实操

之前创建的 `fruit_v` 视图直接使用了 `fruit` 表中的列名称，如果我想要修改成自定义的名称，可以使用如下语句（`CREATE OR REPLACE` 语法完成修改）：

```
-- 自定义视图列名称
mysql> CREATE OR REPLACE VIEW `imooc_mysql`.`fruit_v`(`name_v`, `price_v`) AS SELECT name, price FROM `imooc_mysql`.`fruit`;
Query OK, 0 rows affected (0.04 sec)

-- 验证修改视图定义是否成功
mysql> SELECT * FROM fruit_v;
+--------+---------+
| name_v | price_v |
+--------+---------+
| 苹果   |       5 |
| 香蕉   |       6 |
| 大枣   |       7 |
| 葡萄   |       8 |
+--------+---------+
4 rows in set (0.00 sec)
```

对于所有的水果订单，运营人员想要在原个数（count）之上免费送出一个，可以这样去修改视图（ALTER 语法完成修改，且做了格式化处理）：

```
-- 修改视图的定义
mysql> ALTER VIEW `imooc_mysql`.`fruit_order_v` (
    ->     name ,
    ->     count ,
    ->     price ,
    ->     total
    ->   ) AS SELECT
    ->     name,
    ->     count + 1,
    ->     price,
    ->     count * price
    ->   FROM
    ->     fruit,
    ->     fruit_order
    ->   WHERE
    ->     fruit_id=fruit.id;
Query OK, 0 rows affected (0.03 sec)

-- 验证修改视图定义是否成功
mysql> SELECT * FROM fruit_order_v;
+--------+-------+-------+-------+
| name   | count | price | total |
+--------+-------+-------+-------+
| 苹果   |    13 |     5 |    60 |
| 香蕉   |     9 |     6 |    48 |
| 大枣   |     8 |     7 |    49 |
| 葡萄   |    16 |     8 |   120 |
+--------+-------+-------+-------+
4 rows in set (0.00 sec)
```

# 5. 更新视图

虽然视图是一个虚拟表，但是 MySQL 仍然也允许我们对视图进行插入、更新和删除（但是，强烈不建议这么做）。但是，这里会存在两个问题：更改视图数据，会影响到母表的数据吗？视图可能只涉及母表的部分列，更改会受到限制吗？好的，带着这样的两个问题，我们一起来看看对视图的更改吧。

### 5.1 更新视图实操

对于视图的插入、更新和删除操作，与数据表的操作过程是一致的。关于这些过程是否会对母表数据产生影响的问题，我们以对 fruit_v 视图的操作来验证。如下所示：

```
-- 向 fruit_v 视图中插入数据
mysql> INSERT INTO `fruit_v`(`name_v`, `price_v`) VALUES('榴莲', 9);
Query OK, 1 row affected (0.02 sec)

-- 母表 fruit 中多了一条数据（视图插入的），且 extra 字段是 NULL
mysql> SELECT * FROM fruit;
+----+--------+-------+-------+
| id | name   | price | extra |
+----+--------+-------+-------+
| 1 | 苹果  |     5 | a     |
| 2 | 香蕉  |     6 | b     |
| 3 | 大枣  |     7 | c     |
| 4 | 葡萄  |     8 | d     |
| 5 | 榴莲  |     9 | NULL  |
+----+--------+-------+-------+
5 rows in set (0.00 sec)

-- 更新 fruit_v 视图中的数据
mysql> UPDATE fruit_v SET price_v = price_v + 10 WHERE name_v = '苹果';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

-- 母表中对应的数据 price 由5变成了15
mysql> SELECT * FROM fruit WHERE name = '苹果';
+----+--------+-------+-------+
| id | name   | price | extra |
+----+--------+-------+-------+
| 1 | 苹果  |    15 | a     |
+----+--------+-------+-------+
1 row in set (0.00 sec)

-- 删除 fruit_v 视图中的数据
mysql> DELETE FROM fruit_v WHERE name_v = '榴莲';
Query OK, 1 row affected (0.01 sec)

-- 母表中对应的数据同样被删除
mysql> SELECT * FROM fruit;
+----+--------+-------+-------+
| id | name   | price | extra |
+----+--------+-------+-------+
| 1 | 苹果  |    15 | a     |
| 2 | 香蕉  |     6 | b     |
| 3 | 大枣  |     7 | c     |
| 4 | 葡萄  |     8 | d     |
+----+--------+-------+-------+
4 rows in set (0.00 sec)
```

开头中提到的第一个问题已经得到了答案：更改视图数据，其对应母表中的数据一样会发生变化，即这种变化是同步的。那么，我们再尝试去更改 fruit_order_v 视图数据：

```
-- 插入数据失败
mysql> INSERT INTO `fruit_order_v`(`name`, `count`, `price`, `total`) VALUES('榴莲', 2, 10, 20);
ERROR 1348 (HY000): Column 'count' is not updatable

-- 更新 price 字段成功
mysql> UPDATE fruit_order_v SET price = price + 1;
Query OK, 4 rows affected (0.03 sec)
Rows matched: 4  Changed: 4  Warnings: 0

-- 更新 total 字段失败
mysql> UPDATE fruit_order_v SET total = total + 1;
ERROR 1348 (HY000): Column 'total' is not updatable

-- 删除数据失败
mysql> DELETE FROM fruit_order_v WHERE name = '苹果';
ERROR 1395 (HY000): Can not delete from join view 'imooc_mysql.fruit_order_v'
```

可以看到，更改视图数据的大多数语句都失败了，这也就验证了 MySQL 对于视图的更改是有限制的（可以读一读报错信息，思考下为什么 MySQL 不允许这样做）。

#### 5.2 更新视图的限制

刚刚已经看到 MySQL 对视图的更新其实是有限制的，也就是说，在满足或者不满足某些条件时，视图的更新不被允许。下面，我来对不能更新视图的情况作出总结：

- 视图中并未包含母表中定义为 NOT NULL 的列
- 定义视图的 SELECT 语句（字段列表）中使用了数学表达式
- 定义视图的 SELECT 语句（字段列表）中使用了聚合函数
- 定义视图的 SELECT 语句中使用了 DISTINCT、UNION、GROUP BY、HAVING 子句

根据我这里描述的条件限制，你可以尝试下创建“有条件”的视图，并对之进行修改。验证这些限制的同时，也加深对视图的理解。

## 6. 删除视图

由于某些原因，视图不再需要了，很简单，将它删除就好了。MySQL 提供了 `DROP VIEW` 的语法，可以同时删除一个或者多个视图。如下所示：

```
DROP VIEW [IF EXISTS]
  view_name [, view_name] ...
  [RESTRICT | CASCADE]
```

关于语法的解释如下：

- DROP VIEW 能够删除一个或多个视图，但是，必须要有每个视图的 DROP 权限
- IF EXISTS 用于防止因视图不存在而提示出错
- 如果给定了 RESTRICT 和 CASCADE，将解析并忽略它们

下面，我们就来删除在本节中创建的两个视图，并验证删除成功。执行如下 SQL 语句：

```
-- 删除视图 fruit_v 和 fruit_order_v
mysql> DROP VIEW IF EXISTS fruit_v, fruit_order_v;
Query OK, 0 rows affected (0.01 sec)

-- 验证 fruit_v 被删除
mysql> SELECT * FROM fruit_v;
ERROR 1146 (42S02): Table 'imooc_mysql.fruit_v' doesn't exist

-- 验证 fruit_order_v 被删除
mysql> SELECT * FROM fruit_order_v;
ERROR 1146 (42S02): Table 'imooc_mysql.fruit_order_v' doesn't exist
```

## 7. 总结

在合适的场景或需求下，视图可以大幅降低数据查询的复杂度，且能够保证数据的安全，是 MySQL 中非常重要的知识点。但是，通过本节对视图的介绍，可以看出，理解和使用视图并没有很大的难度。用心学习，不断尝试，你一定可以把视图掌握的很好。

## 8. 问题

你能说出视图的适用场景吗（可以提出一些适合使用视图的需求）？

你能总结出视图与数据表之间的区别与联系吗 ？

创建视图，并尝试对视图进行 CRUD 以及修改结构操作 ？

你觉得删除视图之后，它所对应的母表会受到什么影响呢 ？

## 9. 参考资料

《高性能 MySQL（第三版）》

MySQL 官方文档：CREATE VIEW Statement

MySQL 官方文档：The INFORMATION_SCHEMA VIEWS Table

MySQL 官方文档：DROP VIEW Statement

MySQL 官方文档：Updatable and Insertable Views

MySQL 官方文档：Using Views

}

15 认识日志系统，掌握系统运行过程

17 分区表是什么，又该怎么使用呢？