



下载APP



14 | 视图：如何简化查询？

2021-04-10 朱晓峰

MySQL 必知必会

[进入课程 >](#)**讲述：朱晓峰**

时长 11:35 大小 10.62M



你好，我是朱晓峰。今天，我们来聊一聊视图。

视图是一种虚拟表，我们可以把一段查询语句作为视图存储在数据库中，在需要的时候，可以把视图看做一个表，对里面的数据进行查询。

举个小例子，在学校的信息系统里面，为了减少冗余数据，学生档案（包括姓名、年龄等）和考试成绩（包括考试时间、科目、分数等）是分别存放在不同的数据表里面的，但是，我们经常需要查询学生的考试成绩（包括学生姓名、科目、分数）。这个时候，我们就可以把查询学生考试成绩的这个关联查询，用视图的形式保存起来。这样一来，我们不仅可以从视图中直接查询学生考试成绩，让查询变得简单，而且，视图没有实际存储数据，还避免了数据存储过程中可能产生的冗余，提高了存储的效率。




今天，我就结合超市的项目，来具体讲解一下怎么创建和操作视图，来帮助你提高查询效率。

视图的创建及其好处

首先，我们来学习下创建视图的方法，以及使用视图的一些好处。

创建视图的语法结构：

 复制代码

```
1 CREATE [OR REPLACE]
2 VIEW 视图名称 [(字段列表)]
3 AS 查询语句
```

现在，假设我们要查询一下商品的每日销售明细，这就要从销售流水表（demo.trans）和商品信息表（demo.goodsmaster）中获取到销售数据和对应的商品信息数据。

销售流水表包含流水单号、商品编号、销售数量、销售金额和交易时间等信息：

transno (流水单号)	itemnumber (商品编号)	salesquantity (销售数量)	salesvalue (销售金额)	transdate (交易时间)
3456	1	1	89	2020-12-01
3456	2	1	5	2020-12-01
3457	3	2	20	2020-12-02

商品信息表包含商品编号、条码、名称和售价等信息：

itemnumber (商品编号)	barcode (条码)	goodsname (商品名称)	salesprice (售价)
1	0001	书	89
2	0002	笔	5
3	0003	胶水	10

在不使用视图的情况下，我们可以通过对销售流水表和商品信息表进行关联查询，得到每天商品销售统计的结果，包括销售日期、商品名称、每天销售数量的合计和每天销售金额的合计，如下所示：

[复制代码](#)

```
1 mysql> SELECT
2 -> a.transdate,
3 -> a.itemnumber,
4 -> b.goodsname,
5 -> SUM(a.quantity) AS quantity,    -- 统计销售数量
6 -> SUM(a.salesvalue) AS salesvalue -- 统计销售金额
7 -> FROM
8 -> demo.trans AS a
9 -> LEFT JOIN                -- 连接查询
10 -> demo.goodsmaster AS b ON (a.itemnumber = b.itemnumber)
11 -> GROUP BY a.transdate , a.itemnumber;
12 +-----+-----+-----+-----+-----+
13 | transdate | itemnumber | goodsname | quantity | salesvalue |
14 +-----+-----+-----+-----+-----+
15 | 2020-12-01 00:00:00 | 1 | 本 | 1.000 | 89.00 |
16 | 2020-12-01 00:00:00 | 2 | 笔 | 1.000 | 5.00 |
17 | 2020-12-02 00:00:00 | 3 | 胶水 | 2.000 | 20.00 |
18 +-----+-----+-----+-----+-----+
19 3 rows in set (0.00 sec)
```

在实际项目中，我们发现，每日商品销售查询使用的频次很高，而且经常需要以这个查询的结果为基础，进行更进一步的统计。

举个例子，超市经营者要查一下“每天商品的销售数量和当天库存数量的对比”，如果用 SQL 语句查询，就会比较复杂。历史库存表（demo.inventoryhist）如下所示：

itemnumber (商品编号)	invquantity (库存数量)	invdate (库存日期)
1	100	2020-12-01
2	99	2020-12-01
3	88	2020-12-01
1	149	2020-12-02
2	105	2020-12-02
3	200	2020-12-02


接下来我们的查询步骤会使用到子查询和派生表，很容易理解，你知道含义就行了。

子查询：就是嵌套在另一个查询中的查询。

派生表：如果我们在查询中把子查询的结果作为一个表来使用，这个表就是派生表。

这个查询的具体步骤是：

1. 通过子查询获得单品销售统计的查询结果；
2. 把第一步中的查询结果作为一个派生表，跟历史库存表进行连接，查询获得包括销售日期、商品名称、销售数量和历史库存数量在内的最终结果。

 复制代码

```
1 mysql> SELECT
2   -> a.transdate,
3   -> a.itemnumber,
4   -> a.goodsname,
5   -> a.quantity,          -- 获取单品销售数量
6   -> b.invquantity       -- 获取历史库存数量
7   -> FROM
```


```

8  -> (SELECT          -- 子查询, 统计单品销售
9  -> a.transdate,
10 -> a.itemnumber,
11 -> b.goodsname,
12 -> SUM(a.quantity) AS quantity,
13 -> SUM(a.salesvalue) AS salesvalue
14 -> FROM
15 -> demo.trans AS a
16 -> LEFT JOIN demo.goodsmaster AS b ON (a.itemnumber = b.itemnumber)
17 -> GROUP BY a.transdate , a.itemnumber
18 -> ) AS a -- 派生表, 与历史库存进行连接
19 -> LEFT JOIN
20 -> demo.inventoryhist AS b
21 -> ON (a.transdate = b.invdate
22 -> AND a.itemnumber = b.itemnumber);
23 +-----+-----+-----+-----+-----+
24 | transdate | itemnumber | goodsname | quantity | invquantity |
25 +-----+-----+-----+-----+-----+
26 | 2020-12-01 00:00:00 | 1 | 本 | 1.000 | 100.000 |
27 | 2020-12-01 00:00:00 | 2 | 笔 | 1.000 | 99.000 |
28 | 2020-12-02 00:00:00 | 3 | 胶水 | 2.000 | 200.000 |
29 +-----+-----+-----+-----+-----+
30 3 rows in set (0.00 sec)

```

可以看到，这个查询语句是比较复杂的，可读性和可维护性都比较差。那该怎么办呢？其实，针对这种情况，我们就可以使用视图。

我们可以把商品的每日销售统计查询做成一个视图，存储在数据库里，代码如下所示：

 复制代码


```

1  mysql> CREATE VIEW demo.trans_goodsmaster AS -- 创建视图
2  -> SELECT
3  -> a.transdate,
4  -> a.itemnumber,
5  -> b.goodsname,          -- 从商品信息表中获取名称
6  -> SUM(a.quantity) AS quantity,      -- 统计销售数量
7  -> SUM(a.salesvalue) AS salesvalue   -- 统计销售金额
8  -> FROM
9  -> demo.trans AS a
10 -> LEFT JOIN
11 -> demo.goodsmaster AS b ON (a.itemnumber = b.itemnumber) -- 与商品信息表关联
12 -> GROUP BY a.transdate , a.itemnumber; -- 按照销售日期和商品编号分组
13 Query OK, 0 rows affected (0.01 sec)

```

这样一来，我们每次需要查询每日商品销售数据的时候，就可以直接查询视图，不需要再写一个复杂的关联查询语句了。


我们来试试用一个查询语句直接从视图中进行查询：

 复制代码

```
1 mysql> SELECT * -- 直接查询
2 -> FROM demo.trans_goodsmaster; -- 视图
3 +-----+-----+-----+-----+-----+
4 | transdate | itemnumber | goodsname | quantity | salesvalue |
5 +-----+-----+-----+-----+-----+
6 | 2020-12-01 00:00:00 | 1 | 本 | 1.000 | 89.00 |
7 | 2020-12-01 00:00:00 | 2 | 笔 | 1.000 | 5.00 |
8 | 2020-12-02 00:00:00 | 3 | 胶水 | 2.000 | 20.00 |
9 +-----+-----+-----+-----+-----+
10 3 rows in set (0.01 sec)
```

结果显示，这两种查询方式得到的结果是一样的。

如果我们要进一步查询“每日单品销售的数量与当日的库存数量的对比”，就可以把刚刚定义的视图作为一个数据表来使用。我们把它跟历史库存表连接起来，来获取销售数量和历史库存数量。就像下面的代码这样，查询就简单多了：

 复制代码

```
1 mysql> SELECT
2 -> a.transdate, -- 从视图中获取销售日期
3 -> a.itemnumber, -- 从视图中获取商品编号
4 -> a.goodsname, -- 从视图中获取商品名称
5 -> a.quantity, -- 从视图中获取销售数量
6 -> b.invquantity -- 从历史库存表中获取历史库存数量
7 -> FROM
8 -> demo.trans_goodsmaster AS a -- 视图
9 -> LEFT JOIN
10 -> demo.inventoryhist AS b ON (a.transdate = b.invdate
11 -> AND a.itemnumber = b.itemnumber); -- 直接连接库存历史表
12 +-----+-----+-----+-----+-----+
13 | transdate | itemnumber | goodsname | quantity | invquantity |
14 +-----+-----+-----+-----+-----+
15 | 2020-12-01 00:00:00 | 1 | 本 | 1.000 | 100.000 |
16 | 2020-12-01 00:00:00 | 2 | 笔 | 1.000 | 99.000 |
17 | 2020-12-02 00:00:00 | 3 | 胶水 | 2.000 | 200.000 |
18 +-----+-----+-----+-----+-----+
19 3 rows in set (0.00 sec)
```

结果显示，这里的查询结果和我们刚刚使用派生表的查询结果是一样的。但是，**使用视图的查询语句明显简单多了，可读性更好，也更容易维护。**

如何操作视图和视图中的数据？

创建完了视图，我们还经常需要对视图进行一些操作，比如修改、查看和删除视图。同时，我们可能还需要修改视图中的数据。具体咋操作呢？我来介绍下。

如何操作视图？

修改、查看、删除视图的操作比较简单，你只要掌握具体的语法就行了。

修改视图的语法如下所示：

```
1 ALTER VIEW 视图名
2 AS 查询语句；
```

[复制代码](#)

查看视图的语法是：

```
1 查看视图：
2 DESCRIBE 视图名；
```

[复制代码](#)

删除视图要使用 DROP 关键词，具体方法如下：

```
1 删除视图：
2 DROP VIEW 视图名；
```

[复制代码](#)

好了，到这里，对视图的操作我就介绍完了，下面我再讲讲怎么操作视图中的数据。

如何操作视图中的数据？

刚刚说过，视图本身是一个虚拟表，所以，对视图中的数据进行插入、修改和删除操作，实际都是通过对实际数据表的操作来实现的。

1. 在视图中插入数据

我借用刚刚的视图 `demo.view_goodsmaster` 来给你解释下。假设商品信息表中的规格字段 (`specification`) 被删除了，当我们尝试用 `INSERT INTO` 语句向视图中插入一条记录的时候，就会提示错误了：

[复制代码](#)

```
1 mysql> INSERT INTO demo.view_goodsmaster
2 -> (itemnumber,barcode,goodsname,salesprice)
3 -> VALUES
4 -> (5,'0005','测试',100);
5 ERROR 1471 (HY000): The target table view_goodsmaster of the INSERT is not ins
```

这是因为，**只有视图中的字段跟实际数据表中的字段完全一样，MySQL 才允许通过视图插入数据**。刚刚的视图中包含了实际数据表所没有的字段 “`specification`”，所以在插入数据时，系统就会提示错误。

为了解决这个问题，我们来修改一下视图，让它只包含实际数据表中有的字段，也就是商品编号、条码、名称和售价。代码如下：

[复制代码](#)

```
1 mysql> ALTER VIEW demo.view_goodsmaster
2 -> AS
3 -> SELECT itemnumber,barcode,goodsname,salesprice -- 只包含实际表中存在的字段
4 -> FROM demo.goodsmaster
5 -> WHERE salesprice > 50;
6 Query OK, 0 rows affected (0.01 sec)
```

对视图进行修改之后，我们重新尝试向视图中插入一条记录：

[复制代码](#)


```
1 mysql> INSERT INTO demo.view_goodsmaster
2 -> (itemnumber,barcode,goodsname,salesprice)
3 -> VALUES
4 -> (5,'0005','测试',100);
```



```
5 Query OK, 1 row affected (0.02 sec)
```

结果显示，插入成功了。

现在我们来查看一下视图中的数据：

 复制代码

```
1 mysql> SELECT *
2 -> FROM demo.view_goodsmaster;
3 +-----+-----+-----+-----+
4 | itemnumber | barcode | goodsname | salesprice |
5 +-----+-----+-----+-----+
6 | 1 | 0001 | 本 | 89.00 |
7 | 5 | 0005 | 测试 | 100.00 |
8 +-----+-----+-----+-----+
9 2 rows in set (0.01 sec)
```

-- 通过视图插入的数据

结果显示，表中确实包含了我们插入的商品编号是 5 的商品信息。

现在，视图中已经包括了刚才插入的数据，那么，实际数据表中的数据情况又是怎样的呢？我们再来看一下：

 复制代码

```
1 mysql> SELECT *
2 -> FROM demo.goodsmaster;
3 +-----+-----+-----+-----+
4 | itemnumber | barcode | goodsname | salesprice |
5 +-----+-----+-----+-----+
6 | 1 | 0001 | 本 | 89.00 |
7 | 2 | 0002 | 笔 | 5.00 |
8 | 3 | 0003 | 胶水 | 10.00 |
9 | 5 | 0005 | 测试 | 100.00 |
10 +-----+-----+-----+-----+
11 4 rows in set (0.00 sec)
```

-- 通过视图插入的数据

可以看到，实际数据表 `demo.goodsmaster` 中，也已经包含通过视图插入的商品编号是 5 的商品数据了。


2. 删除视图中的数据

我们可以通过 DELETE 语句，删除视图中的数据：

 复制代码

```
1 mysql> DELETE FROM demo.view_goodsmaster -- 直接在视图中删除数据
2 -> WHERE itemnumber = 5;
3 Query OK, 1 row affected (0.02 sec)
```

现在我们来查看视图和实际数据表的内容，会发现商品编号是 5 的商品都已经被删除了。

 复制代码

```
1 mysql> SELECT *
2 -> FROM demo.view_goodsmaster;
3 +-----+-----+-----+-----+
4 | itemnumber | barcode | goodsname | salesprice |
5 +-----+-----+-----+-----+
6 | 1 | 0001 | 本 | 89.00 |
7 +-----+-----+-----+-----+
8 1 row in set (0.00 sec)
9
10 mysql> SELECT *
11 -> FROM demo.goodsmaster;
12 +-----+-----+-----+-----+
13 | itemnumber | barcode | goodsname | salesprice |
14 +-----+-----+-----+-----+
15 | 1 | 0001 | 本 | 89.00 |
16 | 2 | 0002 | 笔 | 5.00 |
17 | 3 | 0003 | 胶水 | 10.00 |
18 +-----+-----+-----+-----+
19 3 rows in set (0.00 sec)
```

-- 视图中已经没有商品编号是5的商品

-- 实际表中也已经没有商品编号是5


3. 修改视图中的数据

我们可以通过 UPDATE 语句对视图中的数据进行修改：

 复制代码

```
1 mysql> UPDATE demo.view_goodsmaster -- 更新视图中的数据
2 -> SET salesprice = 100
3 -> WHERE itemnumber = 1;
4 Query OK, 1 row affected (0.01 sec)
5 Rows matched: 1 Changed: 1 Warnings: 0
```

结果显示，更新成功了。现在我们来查看一下视图和实际数据表，代码如下所示：

 复制代码

```
1 mysql> SELECT *
2 -> FROM demo.view_goodsmaster;
3 +-----+-----+-----+-----+
4 | itemnumber | barcode | goodsname | salesprice |
5 +-----+-----+-----+-----+
6 | 1 | 0001 | 本 | 100.00 | -- 视图中的售价改过了
7 +-----+-----+-----+-----+
8 1 row in set (0.01 sec)
9
10 mysql> SELECT *
11 -> FROM demo.goodsmaster;
12 +-----+-----+-----+-----+
13 | itemnumber | barcode | goodsname | salesprice |
14 +-----+-----+-----+-----+
15 | 1 | 0001 | 本 | 100.00 | -- 实际数据表中的售价也改过了
16 | 2 | 0002 | 笔 | 5.00 |
17 | 3 | 0003 | 胶水 | 10.00 |
18 +-----+-----+-----+-----+
19 3 rows in set (0.00 sec)
```

可以发现，视图和原来的数据表都已经改过来了。

需要注意的是，**我不建议你对视图的数据进行更新操作**，因为 MySQL 允许用比较复杂的 SQL 查询语句来创建视图（比如 SQL 查询语句中使用了分组和聚合函数，或者是 UNION 和 DISTINCT 关键字），所以，要通过对这个结果集的更新来更新实际数据表，有可能不被允许，因为 MySQL 没办法精确定位实际数据表中的记录。就比如刚刚讲到的那个“每日销售统计查询”视图就没办法更改，因为创建视图的 SQL 语句是一个包含了分组函数（GROUP BY）的查询。

视图有哪些优缺点？

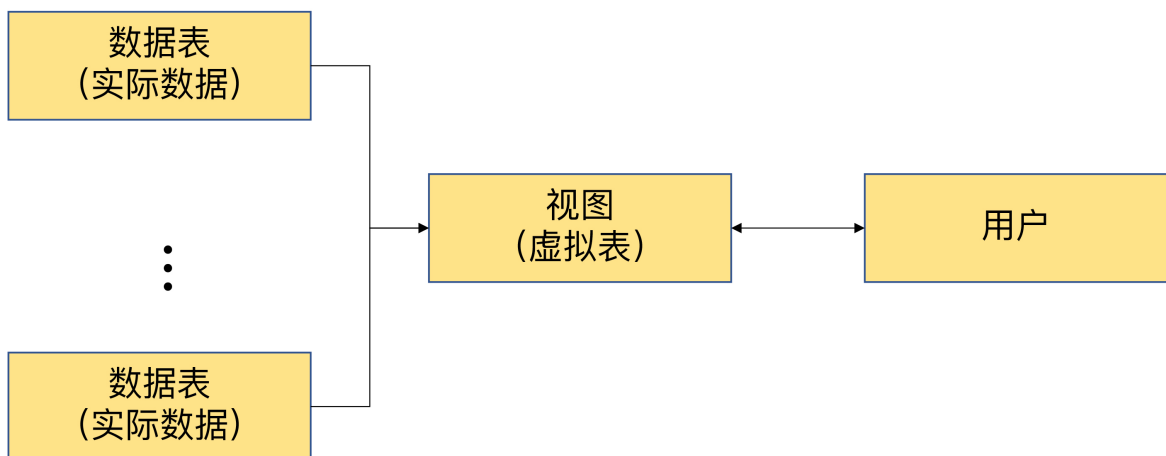
到这里，视图的操作我就讲完了，现在我们把视线拔高一点，来看看视图都有哪些优缺点。只有全面掌握视图的特点，我们才能充分享受它的高效，避免踩坑。

首先，我来介绍下视图的优点。

第一，因为我们可以把视图看成一张表来进行查询，所以在使用视图的时候，我们不用考虑视图本身是如何获取数据的，里面有什么逻辑，包括了多少个表，有哪些关联操作，而是可以直接使用。这样一来，实际上就把查询模块化了，查询变得更加简单，提高了开发和维护的效率。所以，你可以把那些经常会用到的查询和复杂查询的子查询定义成视图，存储到数据库中，这样可以为你以后的使用提供方便。

第二，视图跟实际数据表不一样，它存储的是查询语句。所以，在使用的时候，我们要通过定义视图的查询语句来获取结果集。而视图本身不存储数据，不占用数据存储的资源。

第三，视图具有隔离性。视图相当于在用户和实际的数据表之间加了一层虚拟表。也就是说，**用户不需要查询数据表，可以直接通过视图获取数据表中的信息**。这样既提高了数据表的安全性，同时也通过视图把用户实际需要的信息汇总在了一起，查询起来很轻松。




第四，视图的数据结构相对独立，即便实际数据表的结构发生变化，我们也可以通过修改定义视图的查询语句，让查询结果集里的字段保持不变。这样一来，针对视图的查询就不受实际数据表结构变化的影响了。

这一点不容易理解，我举个小例子来说明一下。

假设我们有一个实际的数据表（demo.goodsmaster），包括商品编号、条码、名称、规格和售价等信息：


itemnumber (商品编号)	barcode (条码)	goodsname (商品名称)	specification (规格)	salesprice (售价)
1	0001	书	16开	89
2	0002	笔	0.5mm	5
3	0003	胶水	水基	10

在这个表的基础上，我们建一个视图，查询所有价格超过 50 元的商品：

 复制代码

```
1 mysql> CREATE VIEW demo.view_goodsmaster AS
2 -> SELECT *
3 -> FROM demo.goodsmaster
4 -> WHERE salesprice > 50;
5 Query OK, 0 rows affected (0.03 sec)
```

接着，我们在这个视图的基础上做一个查询，来验证一下视图的内容：

 复制代码

```
1 mysql> SELECT barcode,goodsname,specification
2 -> FROM demo.view_goodsmaster;
3 +-----+-----+-----+
4 | barcode | goodsname | specification |
5 +-----+-----+-----+
6 | 0001 | 本 | 16开 |
7 +-----+-----+-----+
8 1 row in set (0.00 sec)
```

结果显示，我们得到了商品信息表中售价大于 50 元的商品：本（16 开）。

假设现在我们需要把数据表 demo.goodsmaster 中的字段 “specification” 删掉，就可以用下面的代码：

```
1 mysql> ALTER TABLE demo.goodsmaster DROP COLUMN specification;
2 Query OK, 0 rows affected (0.13 sec)
3 Records: 0 Duplicates: 0 Warnings: 0
```

[复制代码](#)

这样一来，因为少了一个字段，而我们的语句又是直接查询数据表的，代码就会提示错误：

```
1 mysql> SELECT barcode,goodsname,specification
2 -> FROM demo.goodsmaster;
3 ERROR 1054 (42S22): Unknown column 'specification' in 'field list'
```

[复制代码](#)

你看，代码提示字段“specification”不存在。

但是，如果查询的是视图，就可以通过修改视图来规避这个问题。我们可以用下面的代码把刚才的视图修改一下：

```
1 mysql> ALTER VIEW demo.view_goodsmaster
2 -> AS
3 -> SELECT
4 -> itemnumber,
5 -> barcode,
6 -> goodsname,
7 -> '' as specification, -- 由于字段不存在，插入一个长度是0的空字符串作为这个字段的值
8 -> salesprice
9 -> FROM demo.goodsmaster
10 -> WHERE salesprice > 50;
11 Query OK, 0 rows affected (0.02 sec)
```

[复制代码](#)

你看，虽然实际数据表中已经没有字段“specification”了，但是视图中却保留了这个字段，而且字段值始终是空字符串。所以，我们不用修改原有视图的查询语句，它也会正常运行。下面的代码查询的结果中，就包括了实际数据表没有的字段“specification”。

```
1 mysql> SELECT barcode,goodsname,specification
2 -> FROM demo.view_goodsmaster;
3 +-----+-----+-----+
```

[复制代码](#)

```
4 | barcode | goodsname | specification |
5 |-----+-----+-----+
6 | 0001 | 本 | |
7 |-----+-----+-----+
8 | 1 row in set (0.00 sec)
```

结果显示，运行成功了。这个视图查询，就没有受到实际数据表中删除字段的影响。

看到这儿，你可能会说，视图有这么多好处，那我以后都用视图可以吗？其实不是的，视图也有自身的不足。

如果我们在实际数据表的基础上创建了视图，那么，**如果实际数据表的结构变更了，我们就需要及时对相关的视图进行相应的维护**。特别是当视图是由视图生成的时候，维护会变得比较复杂。因为创建视图的 SQL 查询可能会对字段重命名，也可能包含复杂的逻辑，这些都会增加维护的成本。

所以，在创建视图的时候，你要结合实际项目需求，综合考虑视图的优点和不足，这样才能正确使用视图，使系统整体达到最优。

为了方便你掌握，我用一张图来汇总下视图的优缺点：

优点	缺点
视图可以简化查询	数据表的变更，需要及时对视图进行维护，特别是当视图是由视图生成的时候，维护会变得比较复杂
视图不保存数据，不占用数据存储的空间	
视图具有隔离性。视图相当于在用户和实际的数据表之间加了一层。用户不需要直接访问数据表，提高了数据表的安全性，也方便了用户查询	
数据结构相对独立，即便实际表结构产生变化，也可以通过修改视图的 SQL 语句，使用户不受影响	

总结

今天，我给你介绍了简化查询的工具：视图，我们学习了创建视图、操作视图和视图中的数据的方法以及视图的优缺点。你要重点掌握操作的语法结构。

最后，我还是想提醒你一下，虽然可以更新视图数据，但总的来说，视图作为虚拟表，主要用于方便查询。我不建议你更新视图的数据，因为对视图数据的更改，都是通过对实际数据表里数据的操作来完成的，而且有很多限制条件。

视图虽然有很多优点。但是在创建视图、简化查询的同时，也要考虑到视图太多而导致的数据库维护成本的问题。

视图不是越多越好，特别是嵌套的视图（就是在视图的基础上创建视图），我不建议你使用，因为逻辑复杂，可读性不好，容易变成系统的潜在隐患。

思考题

假设某公园售票系统包括门票信息表和类别信息表，这两个表之间通过类别编号相关联。

门票信息表包含门票编号、名称、类别编号和剩余数量等信息。

id (门票编号)	tname (名称)	typeid (类别编号)	balance (剩余数量)
1	入园门票	1	100
2	收费场馆A门票	2	70
3	收费场馆B门票	2	50

类别信息表包含类别编号、开门时间和闭馆时间。

typeid (类别编号)	opentime (开门时间)	closetime (结束时间)
1	9:00	17:00
2	10:00	14:00

请编写一个视图，视图返回的结果集包括：当前时间可以卖的门票名称和剩余数量（说明：开门前 30 分钟开始售票，结束前 30 分钟停止售票）。

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你把它分享给你的朋友或同事，我们下节课见。

提建议

更多课程推荐

深入浅出计算机组成原理

带你掌握计算机体系全貌

徐文浩

bothub 创始人



涨价倒计时 🕒

今日订阅 **¥89**，5月12日涨价至 **¥199**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 13 | 临时表：复杂查询，如何保存中间结果？

下一篇 15 | 存储过程：如何提高程序的性能和安全性？

精选留言 (6)

💬 写留言



朱晓峰 置顶

2021-04-21

你好，我是朱晓峰，下面我就来公布一下上节课思考题的答案：

上节课，我们学习了临时表。下面是思考题的答案：

第一步，先计算门店销售合计...

展开 ▾



不学完不改名 神

2021-04-11

临时表针对当前会话生效，它把数据缓存在内存或磁盘；

而视图仅仅是SQL，每次执行Query时获得原始表的数据。

视图操作数据对应实际的Table，而临时表则是内存或磁盘上的一份拷贝。

...

展开 ▾



lesserror

2021-04-12

在公司下班后用了—个多番茄种学完了这节的内容。要想保持工作之外挤出时间来学习，必须要让自己保持注意力高度集中。

老师「视图」—讲的内容很清晰，这个专栏每—讲的内容都算很具体了。赞—个。

...

展开 ▾



末日, 成欢

2021-04-27

视图会用的索引吗

展开 ▾



Harry

2021-04-11

使用视图后，代码的可读性更好，也更容易维护了。

不建议直接通过视图插入、修改和删除数据，后期维护会出现不可预料的麻烦。

...

展开 ▾



bearlu

2021-04-10

老师，视图和临时表有什么区别？

展开 ▾



