摘要：本文主要讲了MySql中视图的定义、原理和如何使用、创建、删除等

# 一． 视图概述

视图是一个虚拟表，其内容由查询定义。

同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在[数据库](http://www.2cto.com/database/" \t "http://www.2cto.com/database/201508/_blank)中以存储的数据值集形式存在。行和列数据来自

由定义视图的查询所引用的表，并且在引用视图时动态生成。

对其中所引用的基础表来说，视图的作用类似于筛选。定义视图的筛选可以来自当前或其它数据库的一个或多个表，或者其它视图。通过视图进行查询没有任何限制，通过它们进行数据修改时的限制也很少。  
**视图是存储在数据库中的查询的SQL 语句**，它主要出于两种原因：

1.安全原因， 视图可以隐藏一些数据，如：社会保险基金表，可以用视图只显示姓名，地址，而不显示社会保险号和工资数等，

另一原因是可使复杂的查询易于理解和使用。这个视图就像一个“窗口”，从中只能看到你想看的数据列。这意味着你可以在这个视图上使用SELECT \*，而你看到的将是你在视图定义里给出的那些数据列：

既然视图的定义是基于基本表的，哪为什么还要定义视图呢？这是因为合理地使用视图能够带来许多好处：  
1、 视图能简化用户操作  
视图机制使用户可以将注意力集中在所关心地数据上。如果这些数据不是直接来自基本表，则可以通过定义视图，使数据库看起来结构简单、清晰，并且可以简化用户的的数据查询操作。例如，那些定义了若干张表连接的视图，就将表与表之间的连接操作对用户隐藏起来了。换句话说，用户所作的只是对一个虚表的简单查询，而这个虚表是怎样得来的，用户无需了解。

2、 视图使用户能以多种角度看待同一数据  
视图机制能使不同的用户以不同的方式看待同一数据，当许多不同种类的用户共享同一个数据库时，这种灵活性是非常必要的。

3、 视图对重构数据库提供了一定程度的逻辑独立性  
数据的物理独立性是指用户的应用程序不依赖于数据库的物理结构。数据的逻辑独立性是指当数据库重构造时，如增加新的关系或对原有的关系增加新的字段，用户的应用程序不会受影响。层次数据库和网状数据库一般能较好地支持数据的物理独立性，而对于逻辑独立性则不能完全的支持。

在关许数据库中，数据库的重构造往往是不可避免的。重构数据库最常见的是将一个基本表“垂直”地分成多个基本表。例如：将学生关系Student（Sno，Sname，Ssex，Sage，Sdept），  
分为SX（Sno，Sname，Sage）和SY（Sno，Ssex，Sdept）两个关系。这时原表Student为SX表和SY表自然连接的结果。如果建立一个视图Student：

CREATE VIEW Student（Sno，Sname，Ssex，Sage，Sdept）AS SELECT SX.Sno，SX.Sname，SY.Ssex，SX.Sage，SY.Sdept FROM SX，SY WHERE SX.Sno=SY.Sno；

这样尽管数据库的逻辑结构改变了（变为SX和SY两个表了），但应用程序不必修改，因为新建立的视图定义为用户原来的关系，使用户的外模式保持不变，用户的应用程序通过视图仍然能够查找数据。  
当然，视图只能在一定程度上提供数据的逻辑独立，比如由于视图的更新是有条件的，因此应用程序中修改数据的语句可能仍会因为基本表构造的改变而改变。

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](http://www.2cto.com/os/" \t "http://www.2cto.com/database/201508/_blank)时，对不同的用户定义不同的视图，使机密数据不出现在不应该看到这些数据的用户视图上。这样视图机制就自动提供了对机密数据的安全保护功能。例如，Student表涉及全校15个院系学生数据，可以在其上定义15个视图，每个视图只包含一个院系的学生数据，并只允许每个院系的主任查询和修改本原系学生视图。

5、适当的利用视图可以更清晰地表达查询  
例如经常需要执行这样的查询“**对每个学生找出他获得最高成绩的课程号**”。可以先定义一个视图，求出每个同学获得的最高成绩：

CREATE VIEW VMGRADE

AS

SELECT Sno，MAX(Grade) Mgrade

FROM SC

GROUP BY Sno

然后用如下的查询语句完成查询：

SELECT SC.Sno，Cno FROM SC，VMGRADE

WHERE SC.Sno = VMGRADE.Sno AND SC.Grade = VMGRADE.Mgrade；

# 二、数据准备

1、员工表

CREATE TABLE t\_employee(

ID INT PRIMARY KEY AUTO\_INCREMENT,

NAME CHAR(30) NOT NULL,

SEX CHAR(2) NOT NULL,

AGE INT NOT NULL,

DEPARTMENT CHAR(10) NOT NULL,

SALARY INT NOT NULL,

HOME CHAR(30),

MARRY CHAR(2) NOT NULL DEFAULT '否',

HOBBY CHAR(30)

);

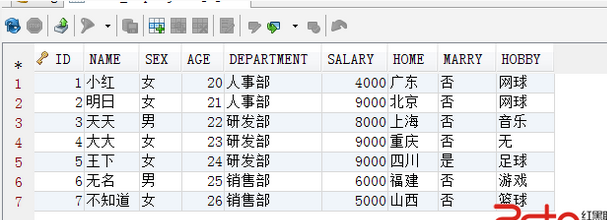
插入数据：

INSERT INTO learning.t\_employee(ID, NAME, SEX, AGE,DEPARTMENT, SALARY, HOME, MARRY, HOBBY) VALUES(NULL,'小红','女',20,'人事部','4000','广东','否','网球');

INSERT INTO learning.t\_employee(ID, NAME, SEX, AGE,DEPARTMENT, SALARY, HOME, MARRY, HOBBY) VALUES(NULL,'明日','女',21,'人事部','9000','北京','否','网球');

INSERT INTO learning.t\_employee(ID, NAME, SEX, AGE,DEPARTMENT, SALARY, HOME, MARRY, HOBBY) VALUES(NULL,'天天','男',22,'研发部','8000','上海','否','音乐');

插入的结果：



然后再定义一张员工信息表：

create TABLE t\_employee\_detail(

ID INT PRIMARY KEY,

POS CHAR(10) NOT NULL,

EXPERENCE CHAR(10) NOT NULL,

CONSTRAINT `FK\_ID` FOREIGN KEY(ID) REFERENCES t\_employee(ID)

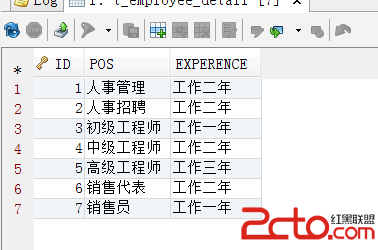
)

插入如下：

INSERT INTO t\_employee\_detail(ID,POS,EXPERENCE) VALUES(1,'人事管理','工作二年');

INSERT INTO t\_employee\_detail(ID,POS,EXPERENCE) VALUES(2,'人事招聘','工作二年');

内容：



# 使用案例

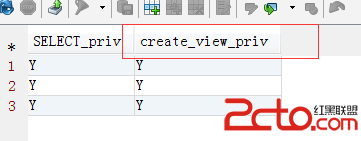
# 1. 语法 CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] VIEW [db\_name.]view\_name [(column\_list)] AS select\_statement [WITH [CASCADED | LOCAL] CHECK OPTION]

通过该语句可以创建视图，**若给定了[OR REPLACE]，则表示当已具有同名的视图时，将覆盖原视图**。select\_statement是一个查询语句，这个查询语句可从表或其它的视图中查 询。视图属于数据库，因此需要指定数据库的名称，若未指定时，表示在当前的数据库创建新视图。  
表和数据库共享数据库中相同的名称空间，因此，数据库不能包含相同名称的表和视图，并且，视图的列名也不能重复。

在创建视图前应先看看是否有权限：

SELECT SELECT\_priv,create\_view\_priv from [mysql](http://www.2cto.com/database/MySQL/" \t "http://www.2cto.com/database/201508/_blank).user WHERE user='root'

Y表示有创建的权限



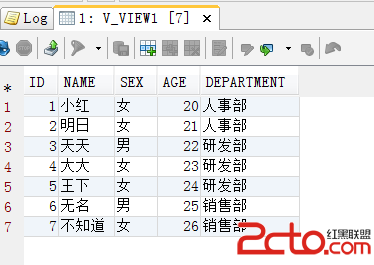
2、单表上创建视图

在员工表是创建视图

CREATE VIEW V\_VIEW1(ID, NAME, SEX, AGE,DEPARTMENT) AS SELECT ID, NAME, SEX, AGE,DEPARTMENT FROM learning.t\_employee;

然后是显示内容：

SELECT \* FROM V\_VIEW1



3、多表上创建视图

CREATE VIEW V\_VIEW2(ID, NAME, SEX, AGE,DEPARTMENT,POS,EXPERENCE)

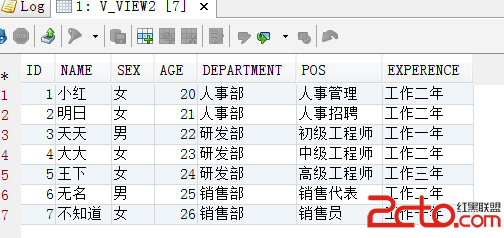
AS SELECT a.ID, a.NAME, a.SEX, a.AGE,a.DEPARTMENT,b.POS,b.EXPERENCE

FROM learning.t\_employee a,learning.t\_employee\_detail b

WHERE a.ID=b.ID;

显示结果

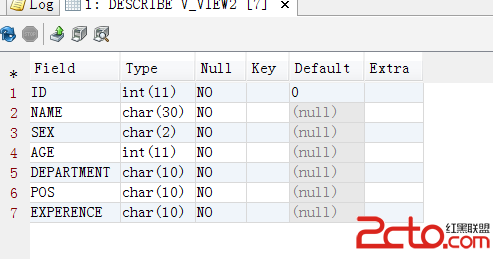
SELECT \*FROM V\_VIEW2



4、查看视图

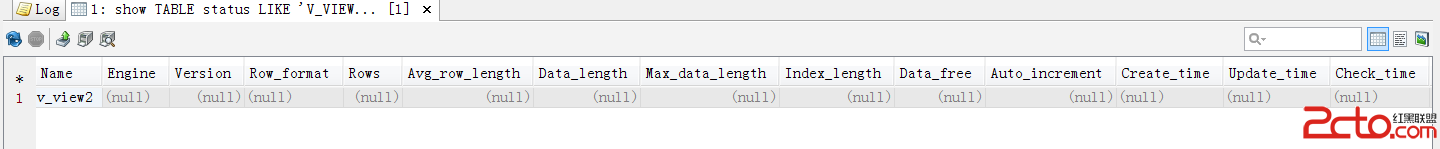
（1）DESCRIBE 命令

DESCRIBE V\_VIEW2



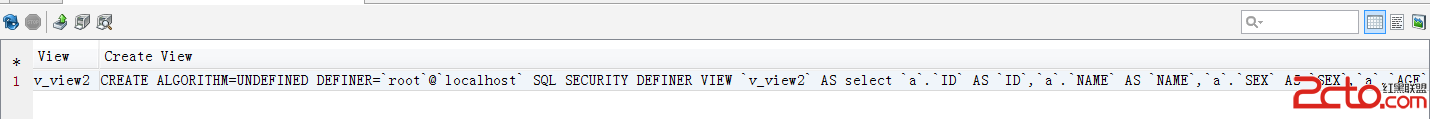
（2）SHOW TABLE STATUS：显示视图信息

show TABLE status LIKE 'V\_VIEW2'



SHOW CREATE view命令：查询视图的定义

show CREATE view V\_VIEW2



5、修改视图

(1)CREATE OR REPLACE命令

CREATE OR REPLACE VIEW V\_VIEW1(ID, NAME, SEX)

AS SELECT ID, NAME, SEX FROM learning.t\_employee;



(2) ALTER 命令

ALTER VIEW V\_VIEW1(ID, NAME) AS SELECT ID, NAME FROM learning.t\_employee;

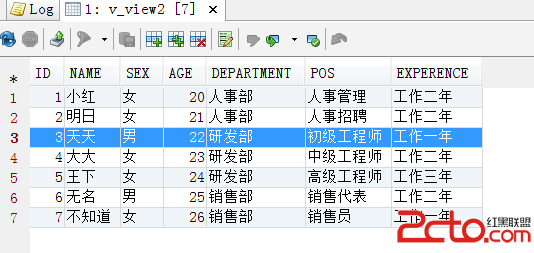
SELECT \* FROM learning.v\_view1



6、更新视图

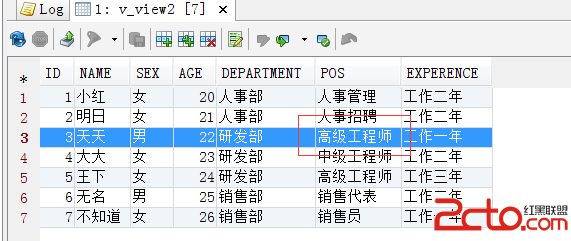
在MySQL中，更新视图是指通过视图来插入（INSERT）、更新（UPDATE）和删除（DELETE）表中的数据。因为视图是一个虚拟表，其中没有数据，所以通过视图更新时，都是转换到基本表来更新。  
更新视图时，只能更新权限范围内的数据。超出了范围，就不能更新。

更新前：



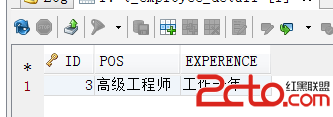
更新后：

UPDATE V\_VIEW2 SET POS='高级工程师' WHERE NAME='天天'



对应 的真实表上的数据也发生改变了

SELECT \* FROM learning.t\_employee\_detail WHERE t\_employee\_detail.ID=3



不可更新的视图：  
某些视图是可更新的。也就是说，可以在诸如UPDATE、DELETE或INSERT等语句中使用它们，以更新基表的内容。对于可更新的视图，在视图中的行和基表中的行之间必须具有一对一的关系。还有一些特定的其他结构，这类结构会使得视图不可更新。更具体地讲，如果视图包含下述结构中的任何一种，那么它就是不可更新的：

· 聚合函数（SUM(), MIN(), MAX(), COUNT()等）。  
· DISTINCT  
· GROUP BY  
· HAVING  
· UNION或UNION ALL  
· 位于选择列表中的子查询  
· Join  
· FROM子句中的不可更新视图  
· WHERE子句中的子查询，引用FROM子句中的表。  
· 仅引用文字值（在该情况下，没有要更新的基本表）。  
· ALGORITHM = TEMPTABLE（使用临时表总会使视图成为不可更新的）。

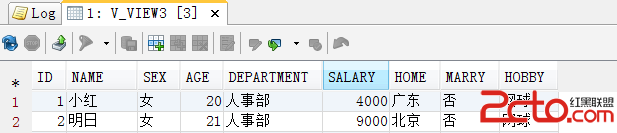
注意  
视图中虽然可以更新数据，但是有很多的限制。一般情况下，最好将视图作为查询数据的虚拟表，而不要通过视图更新数据。因为，使用视图更新数据时，如果没有全面考虑在视图中更新数据的限制，就可能会造成数据更新失败。

CASCADED和LOCAL能不能决定视图是否能更新？  
WITH[CASCADED|LOCAL] CHECK OPTION能不能决定视图是否能更新？这两个参数的基本定义如下：  
LOCAL参数表示更新视图时只要满足该视图本身定义的条件即可。  
CASCADED参数表示更新视图时需要满足所有相关视图和表的条件。没有指明时，该参数为默认值。

With check option的用法：  
（with check option对于没有where条件的视图不起作用的）

CREATE VIEW V\_VIEW3(ID, NAME,SEX,AGE,DEPARTMENT,SALARY, HOME, MARRY, HOBBY) AS SELECT ID, NAME, SEX,AGE,DEPARTMENT,SALARY,HOME,MARRY,HOBBY FROM learning.t\_employee WHERE DEPARTMENT='人事部' WITH LOCAL CHECK OPTION;

表示只限定插入部门为人事部的人。

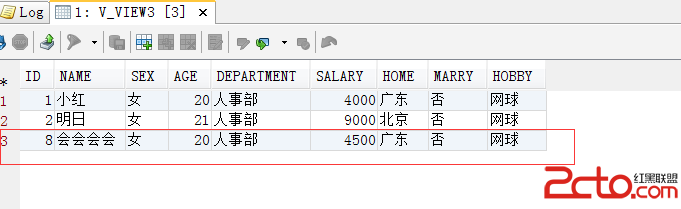


然后插入一条：

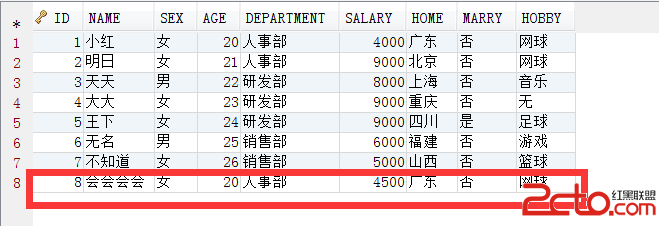
INSERT INTO learning.V\_VIEW3(ID, NAME, SEX, AGE,DEPARTMENT, SALARY, HOME, MARRY, HOBBY) VALUES(NULL,'会会会会','女',20,'人事部','4500','广东','否','网球');

看下结果：

SELECT \* FROM learning.V\_VIEW3

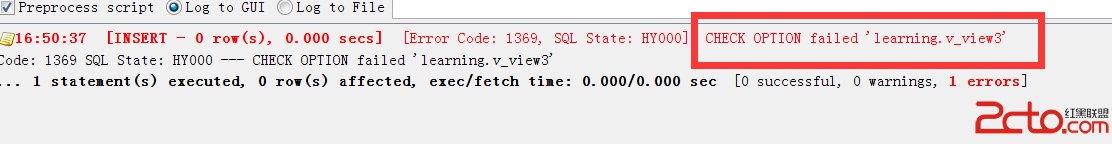


同时看真实表中的数据：



再来插入一条：

INSERT INTO learning.V\_VIEW3(ID, NAME, SEX, AGE,DEPARTMENT, SALARY, HOME, MARRY, HOBBY) VALUES(NULL,'qqqqvasvas','女',20,'研发部','4500','上海','否','网球');



结果显示插入失败  
对于with check option用法，总结如下：  
通过有with check option选项的视图操作基表(只是面对单表，对连接多表的视图正在寻找答案)，有以下结论： 插入后的数据，通过视图能够查询出来就符合WITH CHECK OPTION 否则就不符合；  
首先视图只操作它可以查询出来的数据，对于它查询不出的数据，即使基表有，也不可以通过视图来操作。  
1.对于update,有with check option，要保证update后，数据要被视图查询出来  
2.对于delete,有无with check option都一样  
4.对于insert,有with check option，要保证insert后，数据要被视图查询出来  
对于没有where 子句的视图，使用with check option是多余的

7、删除视图

DROP VIEW IF EXISTS 视图名