

第3章 SQL之数据更新

大纲

- SQL概述
- 学生-课程数据库
- 数据定义
- 数据查询
- **数据更新**
- 空值的处理
- 视图
- 本章小结

数据更新

- **插入数据**
- 修改数据
- 删除数据

插入数据

■ 语法格式:

INSERT INTO <表名>[(<属性列1>[,<属性列2>...]) **VALUES** (<常量1>[,<常量2>]...);

- 功能：将新元组插入到指定表中

INTO子句	VALUES子句
<ul style="list-style-type: none">• 指定要插入数据的表名及属性列• 属性列的顺序可与表定义中的顺序不一致• 没有指定属性列：表示要插入的是一条完整的元组，且属性列属性与表定义中的顺序一致• 指定部分属性列：插入的元组在其余属性列上取空值	<ul style="list-style-type: none">• 提供的值必须与INTO子句匹配<ul style="list-style-type: none">• 值的个数• 值的类型

— 两种插入数据的两种方式

- 插入元组：用于插入新元组
- 插入子查询结果：利用已有数据导出的结果

[例3.69] 将一个新学生元组(学号: 201215128; 姓名:陈冬; 性别:男; 所在系:IS; 年龄:18岁) 插入到Student表中。

```
INSERT INTO Student(Sno,Sname,Ssex,Sdept,Sage)
VALUES ('201215128','陈冬','男','IS',18);
```

[例3.70] 将学生张成民的信息插入到Student表中。

```
INSERT INTO Student
VALUES ('201215126','张成民','男', 18, 'CS');
```

[例3.71] 插入一条选课记录 ('201215128','1')

```
INSERT INTO SC(Sno, Cno)
VALUES ('201215128','1');
```

- 关系数据库管理系统将在新插入记录的Grade列上自动地赋空值或者

```
INSERT INTO SC VALUES ('201215128','1', NULL);
```

■ 插入子查询结果:

INSERT INTO <表名> [(<属性列1>[,<属性列2>...)] 查询;

- 查询中的SELECT子句的目标列必须与INTO子句后面的属性列匹配
 - 值的个数、值的类型

[例3.72] 对每一个系，求学生的平均年龄，并把结果存入数据库。

第一步：建表

```
CREATE TABLE Dept_age
(Sdept CHAR(15), --系名
Avg_age SMALLINT /*学生平均年龄 */
);
```

第二步：插入数据

```
INSERT
INTO Dept_age (Sdept, Avg_age)
SELECT Sdept, AVG(Sage)
FROM Student
GROUP BY Sdept;
```

- 关系数据库管理系统在执行插入语句时会检查所插元组是否破坏表上已定义的完整性规则。
 - 实体完整性
 - 参照完整性
 - 用户定义的完整性
 - NOT NULL约束
 - UNIQUE约束
 - 值域约束

openGauss的INSERT命令插入多行

```
openGauss=# CREATE TABLE
customer_t1
(
c_customer_sk integer,
c_customer_id char(5),
c_first_name char(6),
c_last_name char(8)
);
```

- 如果需要向表中插入多条数据，除了可以多次执行插入一行数据命令，也可以执行一次插入多条数据实现
- 使用此命令可以提升效率(建议)

```
openGauss=# INSERT INTO customer_t1(c_customer_sk, c_customer_id,c_first_name)
VALUES (6885,'maps','Joes'), (4321, 'tpcds','Lily'), (9527,'world','James');
```


数据更新

- 插入数据
- **修改数据**
- 删除数据

修改数据

■ 语句格式:

UPDATE <表名> SET <列名> = <表达式> [, <列名> = <表达式>] ... [WHERE <条件>];

— 功能:

- 修改指定表中满足WHERE子句条件的元组
- SET子句给出<表达式>的值用于取代相应的属性列
- 如果省略WHERE子句, 表示要修改表中的所有元组

■ 数据修改的三种方式:

– 修改某一个元组的值

[例3.73] 将学生201215121的年龄改为22岁。

```
UPDATE Student SET Sage=22 WHERE Sno='201215121' ;
```

– 修改多个元组的值

[例3.74] 将所有学生的年龄都增加1岁。

```
UPDATE Student SET Sage=Sage +1;
```

– 带子查询的修改语句

[例3.75] 将计算机科学系全体学生的成绩置零。

```
UPDATE SC SET Grade=0 WHERE Sno IN (SELECT Sno  
                                     FROM Student  
                                     WHERE Sdept='CS');
```

- 关系数据库管理系统在执行修改语句时会检查修改操作是否破坏表上已定义的完整性规则。
 - 实体完整性
 - 参照完整性
 - 用户定义的完整性
 - NOT NULL约束
 - UNIQUE约束
 - 值域约束

数据更新

- 插入数据
- 修改数据
- **删除数据**

删除数据

- 语句格式:

DELETE FROM <表名> [WHERE <条件>];

- 功能:

- 删除指定表中满足WHERE子句条件的元组。

- WHERE子句

- 指定要删除的元组。
 - 缺省表示要删除表中的全部元组，表的定义仍在字典中。

■ 数据删除的三种方式:

– 删除某一个元组的值

[例3.76] 删除学号为201215128的学生记录

```
DELETE FROM Student WHERE Sno='201215128' ;
```

– 删除多个元组的值

[例3.77] 删除所有的学生选课记录

```
DELETE FROM SC;
```

– 带子查询的删除语句

[例3.78] 删除计算机科学系所有学生的选课记录

```
DELETE FROM SC WHERE Sno IN (SELECT Sno  
                                FROM Student  
                                WHERE Sdept='CS');
```

openGauss之TRUNCATE命令

■ 语法格式:

TRUNCATE TABLE table_name;

- 功能: 清理表数据, TRUNCATE快速地从表中删除所有行

DELETE、DROP和TRUNCATE三者比较

DELETE	DROP	TRUNCATE
<ul style="list-style-type: none">• 删除表中满足条件的所有行• 逐行删除, 每删除一行将在事务日志登记删除记录• 删除内容, 不删除定义, 不释放空间	<ul style="list-style-type: none">• 删除内容和定义, 释放空间	<ul style="list-style-type: none">• 效果同DELETE FROM table_name;• 不扫描表, 按数据页删除, 因而删除速度快, 使用系统资源和事务日志少• 删除内容, 不删除定义, 释放空间

[例3.77]可用TRUNCATE改写: TRUNCATE TABLE SC;

数据更新小结

- 数据更新包括数据的插入、修改和删除
 - 三种操作都**只能在单表上**操作
 - 语法格式上易与多表查询混淆
- 当**修改或删除**操作涉及多张表时，只能通过**子查询完成**
- 一些RDBMS产品，如openGauss、MySQL支持**一条insert语句实现插入多条记录**，而不需要分别执行多条insert语句
- openGauss的truncate删除命令效率比delete更高，因其需要的事务日志资源更少

大纲

- SQL概述
- 学生-课程数据库
- 数据定义
- 数据查询
- 数据更新
- **空值的处理**
- 视图
- 本章小结

空值的处理

- 空值(Null)就是“不知道”或“不存在”或“无意义”的值。
- 一般有以下几种情况：
 - 该属性应该有一个值，但目前不知道它的具体值
 - 该属性不应该有值
 - 由于某种原因不便于填写
- 空值是一个很特殊的值，含有不确定性。对关系运算带来特殊的问题，需要做特殊的处理。

[例 3.79] 向SC表中插入一个元组，学生号是'201215126'，课程号是 '1'，成绩为空.

```
INSERT INTO SC (Sno,Cno,Grade)
VALUES ( '201215126' , '1' ,NULL) ; /*该学生还没有考试成绩，取空值*/
```

或

```
INSERT INTO SC (Sno,Cno)
VALUES ( '201215126' , '1' ) ; /*没有赋值的属性，其值为空值*/
```

[例3.80] 将Student表中学生号为'201215200'的学生所属的系改为空值。

```
UPDATE Student
SET Sdept=NULL WHERE Sno='201215200' ;
```

■ 空值的使用：

- 判断一个属性的值是否为空值，用IS NULL或IS NOT NULL来表示

[例 3.81] 从Student表中找出漏填了数据的学生信息。

```
SELECT  *  
FROM Student  
WHERE Sname IS NULL OR Ssex IS NULL OR Sage IS NULL OR Sdept IS NULL;
```

■ 空值的约束条件：

- 在属性定义(或者域定义)中
 - 有NOT NULL约束条件的不能取空值
 - 加了UNIQUE限制的属性不能取空值 ---教材的这条结论已不完全正确，见openGauss例子
 - 码属性不能取空值

```
CREATE TABLE test(  
    Eno char(10) PRIMARY KEY,  
    Ename varchar(20) UNIQUE);
```

```
db_demo=> create table test(eno char(10) primary key, ename varchar(20) unique);  
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "test_pkey" for table "test"  
NOTICE: CREATE TABLE / UNIQUE will create implicit index "test_ename_key" for table "test"  
CREATE TABLE  
db_demo=> \dt  
  
                List of relations  


| Schema | Name | Type  | Owner  | Storage                          |
|--------|------|-------|--------|----------------------------------|
| public | test | table | wanghj | {orientation=row,compression=no} |

  
(1 row)
```

openGauss例子

```
db_demo=> insert into test values('1',null), ('2',null);  
INSERT 0 2  
db_demo=> select * from test;  


| eno | ename |
|-----|-------|
| 1   |       |
| 2   |       |

  
(2 rows)
```

```
mysql> create table emp(eno char(8) primary key, ename varchar(15) unique);  
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> insert into emp values('1');  
ERROR 1136 (21S01): Column count doesn't match value count at row 1  
mysql> insert into emp(eno) values('1');  
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from emp;
```

eno	ename
1	NULL

```
1 row in set (0.00 sec)
```

```
mysql> insert into emp(eno) values('2');  
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from emp;
```

eno	ename
1	NULL
2	NULL

MySQL例子

■ 空值的算术运算、比较运算和逻辑运算

- 空值与另一个值(包括另一个空值)的算术运算的结果为空值
- 空值与另一个值(包括另一个空值)的比较运算的结果为UNKNOWN
- 有UNKNOWN后，传统二值(TRUE, FALSE)逻辑就扩展成了三值逻辑

真值表

x	y	x AND y	x OR y	NOT x
T	T	T	T	F
T	U	U	T	F
T	F	F	T	F
U	T	U	T	U
U	U	U	U	U
U	F	F	U	U
F	T	F	T	T
F	U	F	U	T
F	F	F	F	T

- T: True
- F: False
- U: Unknown

[例3.82] 找出选修1号课程不及格学生的学号。

```
SELECT Sno
FROM SC
WHERE Grade<60 AND Cno='1';
```

查询结果不包括缺考的学生，
因为他们的Grade的值为
NULL，是不可比较大小的。

[例3.83] 找出选修1号课程不及格学生的学号以及缺考学生的学号。

```
SELECT Sno
FROM SC
WHERE Grade < 60 AND Cno='1'
UNION
SELECT Sno
FROM SC
WHERE Grade IS NULL AND Cno='1';
或者
SELECT Sno
FROM SC
WHERE Cno='1' AND (Grade<60 OR Grade IS NULL);
```

大纲

- SQL概述
- 学生-课程数据库
- 数据定义
- 数据查询
- 数据更新
- 空值的处理
- 视图
- 本章小结

视图

■ 视图的特点

- 虚表，是从一个或几个基本表（或视图）导出的表。
- 只存放视图的定义，不存放视图对应的数据。
- 基表中的数据发生变化，从视图中查询出的数据也随之改变。

视图

- **定义视图**
- 删除视图
- 查询视图
- 更新视图
- 视图的作用

定义(创建)视图

■ 语句格式:

```
CREATE VIEW <视图名>[(<列名> [,<列名>]...)]  
AS <子查询> [WITH CHECK OPTION];
```

视图字段全部省略:

- 由子查询中SELECT目标列中的诸字段组成

指明字段:

- 某个目标列是聚集函数或列表表达式
- 多表连接时选出了几个同名列作为视图的字段
- 需要在视图中为某个列启用新的更合适的名字

— WITH CHECK OPTION子句

- 对视图进行UPDATE, INSERT和DELETE操作时要保证更新、插入或删除的行满足视图定义中的谓词条件 (即子查询中的条件表达式) 。
 - 子查询可以是任意的SELECT语句, 是否可以含有ORDER BY子句和DISTINCT短语, 则决定具体系统的实现。
- 关系数据库管理系统执行CREATE VIEW语句时只是把视图定义存入数据字典, 并不执行其中的SELECT语句。
- 在对视图查询时, 按视图的定义从基本表中将数据查出。

[例3.84] 建立信息系学生的视图。

```
CREATE VIEW IS_Student
AS
SELECT Sno, Sname, Sage
FROM Student
WHERE Sdept= 'IS';
```

[例3.85] 建立信息系学生的视图，并要求进行修改和插入操作时仍需保证该视图只有信息系的学生。

```
CREATE VIEW IS_Student
AS
SELECT Sno, Sname, Sage
FROM Student
WHERE Sdept= 'IS'
WITH CHECK OPTION;
```



定义IS_Student视图时加上了
WITH CHECK OPTION子句，
对该视图进行插入、修改和删
除操作时，RDBMS会自动加上
Sdept= 'IS' 的条件。

视图的构造类型

■ 行列子集视图

- 若一个视图是从单个基本表导出的，并且只是去掉了基本表的某些行和某些列，但保留了主码。如，IS_Student为行列子集视图

■ 基于多个基表的视图

[例3.86] 建立信息系选修了1号课程的学生视图(包括学号、姓名、成绩)。

```
CREATE VIEW IS_S1(Sno, Sname, Grade)
AS
SELECT Student.Sno, Sname, Grade
FROM Student, SC
WHERE Sdept='IS' AND Student.Sno=SC.Sno AND SC.Cno='1';
```

■ 基于视图的视图

[例3.87] 建立信息系选修了1号课程且成绩在90分以上的学生的视图。

```
CREATE VIEW IS_S2
AS
SELECT Sno, Sname, Grade
FROM IS_S1
WHERE Grade >= 90;
```

■ 基于表达式的视图

[例3.88] 定义一个反映学生出生年份的视图。

```
CREATE VIEW BT_S(Sno, Sname, Sbirth)
AS
SELECT Sno, Sname, 2014-Sage
FROM Student;
```



■ 分组视图

[例3.89] 将学生的学号及平均成绩定义为一个视图。

```
CREAT VIEW S_G(Sno, Gavg)
AS
SELECT Sno, AVG(Grade)
FROM SC
GROUP BY Sno;
```

[例3.89] 将Student表中所有女生记录定义为一个视图。

```
CREATE VIEW F_Student(F_Sno, name, sex, age, dept)
AS
SELECT * /*没有不指定属性列*/
FROM Student
WHERE Ssex='女';
```



修改基表Student的结构后, Student表与F_Student视图的映象关系被破坏, 导致该视图不能正确工作

视图

- 定义视图
- **删除视图**
- 查询视图
- 更新视图
- 视图的作用

删除视图

■ 语句格式:

DROP VIEW <视图名> [**CASCADE**];

- 该语句从数据字典中删除指定的视图定义。
- 如果该视图上还导出了其他视图，使用CASCADE级联删除语句，把该视图和由它导出的所有视图一起删除。
- 删除基表时，由该基表导出的所有视图定义都必须显式地使用DROP VIEW语句删除。

[例3.91] 删除视图BT_S和IS_S1。

```
DROP VIEW BT_S;           /*成功执行*/  
DROP VIEW IS_S1;          /*拒绝执行*/  
  
DROP VIEW IS_S1 CASCADE; /*成功执行*/
```

视图

- 定义视图
- 删除视图
- **查询视图**
- 更新视图
- 视图的作用

查询视图

- 用户角度：查询视图与查询基本表相同
- 关系数据库管理系统实现视图查询的方法

- 视图消解法(View Resolution)

有效性检查

转换成等价的对基本表的查询

执行修正后的查询

- 物化视图(Materialized View)

- 物化视图用于预先计算并保存表连接或聚集等耗时较多的操作的结果，这样在执行查询时，就可以避免进行这些耗时的操作，快速得到结果，从而提高查询性能。

物化视图的特点：

- 物理真实存在，故需要占用存储空间；
- 物化视图对应用透明，增加和删除物化视图不会影响应用程序中SQL语句的正确性和有效性；
- 当基本表发生变化时，物化视图也应刷新。

oracle 物化视图参考：<https://blog.csdn.net/yangshangwei/article/details/53328605>

[例3.92] 在信息系学生的视图中找出年龄小于20岁的学生。

```
SELECT Sno,Sage  
FROM IS_Student  
WHERE Sage < 20;
```



```
SELECT Sno,Sage  
FROM Student  
WHERE Sdept='IS' AND Sage<20;
```

[例3.93] 查询选修了1号课程的信息系学生。

```
SELECT IS_Student.Sno, Sname  
FROM IS_Student, SC  
WHERE IS_Student.Sno=SC.Sno AND SC.Cno='1';
```

■ 视图消解法的局限

- 有些情况下，视图消解法不能生成正确的查询

[例3.94] 在S_G视图中查询平均成绩在90分以上的学生学号和平均成绩。

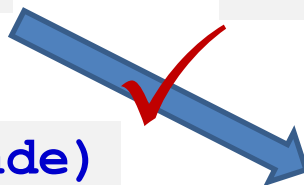
```
SELECT *  
FROM S_G  
WHERE Gavg >= 90;
```



```
SELECT Sno, AVG(Grade)  
FROM SC  
WHERE AVG(Grade) >= 90  
GROUP BY Sno;
```



```
CREATE VIEW S_G(Sno, Gavg)  
AS  
SELECT Sno, AVG(Grade)  
FROM SC  
GROUP BY Sno;
```



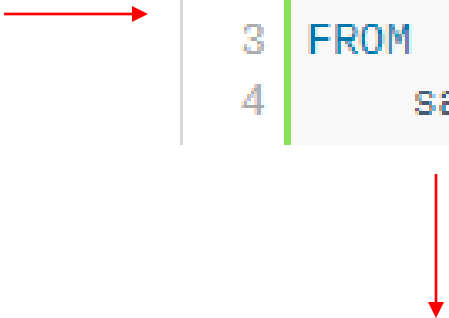
```
SELECT Sno, AVG(Grade)  
FROM SC  
GROUP BY Sno  
HAVING AVG(Grade) >= 90;
```

[例3.94]也可以用如下SQL语句完成。

```
SELECT *  
FROM (SELECT Sno,AVG(Grade)  
      FROM SC  
      GROUP BY Sno) AS S_G(Sno,Gavg)  
WHERE Gavg>=90;
```


oracle视图举例

```
1 CREATE VIEW salesman AS
2 SELECT
3     *
4 FROM
5     employees
6 WHERE
7     job_title = 'Sales Representative';
```



```
1 SELECT
2     *
3 FROM
4     salesman;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE	HIRE_DATE	MANAGER_ID	JOB_TITLE
56	Evie	Harrison	evie.harrison@example.com	011.44.1344.486508	23-NOV-07	46	Sales Representative
57	Scarlett	Gibson	scarlett.gibson@example.com	011.44.1345.429268	30-JAN-04	47	Sales Representative
58	Ruby	Mcdonald	ruby.mcdonald@example.com	011.44.1345.929268	04-MAR-04	47	Sales Representative
59	Chloe	Cruz	chloe.cruz@example.com	011.44.1345.829268	01-AUG-04	47	Sales Representative
60	Isabelle	Marshall	isabelle.marshall@example.com	011.44.1345.729268	10-MAR-05	47	Sales Representative
61	Daisy	Ortiz	daisy.ortiz@example.com	011.44.1345.629268	15-DEC-05	47	Sales Representative
62	Freya	Gomez	freya.gomez@example.com	011.44.1345.529268	03-NOV-06	47	Sales Representative
80	Elizabeth	Dixon	elizabeth.dixon@example.com	011.44.1644.429262	04-JAN-08	50	Sales Representative
64	Florence	Freeman	florence.freeman@example.com	011.44.1346.229268	19-MAR-07	48	Sales Representative
65	Alice	Wells	alice.wells@example.com	011.44.1346.329268	24-JAN-08	48	Sales Representative

```
1 CREATE VIEW salesman_contacts AS
2 SELECT
3     first_name,
4     last_name,
5     email,
6     phone
7 FROM
8     salesman;
```

```
1 SELECT
2     *
3 FROM
4     salesman_contacts;
```

FIRST_NAME	LAST_NAME	EMAIL	PHONE
Evie	Harrison	evie.harrison@example.com	011.44.1344.486508
Scarlett	Gibson	scarlett.gibson@example.com	011.44.1345.429268
Ruby	Mcdonald	ruby.mcdonald@example.com	011.44.1345.929268
Chloe	Cruz	chloe.cruz@example.com	011.44.1345.829268
Isabelle	Marshall	isabelle.marshall@example.com	011.44.1345.729268
Daisy	Ortiz	daisy.ortiz@example.com	011.44.1345.629268
Freya	Gomez	freya.gomez@example.com	011.44.1345.529268
Elizabeth	Dixon	elizabeth.dixon@example.com	011.44.1644.429262
Florence	Freeman	florence.freeman@example.com	011.44.1346.229268
Alice	Wells	alice.wells@example.com	011.44.1346.329268

```
1 DROP VIEW salesman;
```

```
1 SELECT
2     object_name,
3     status
4 FROM
5     user_objects
6 WHERE
7     object_type = 'VIEW'
8     AND object_name = 'SALESMAN_CONTACTS';
```

OBJECT_NAME	STATUS
SALESMAN_CONTACTS	INVALID

```
1 DROP VIEW salesman_contacts;
```

- 因为salesman_contacts视图依赖于salesman视图，所以salesman被删除后它就变得无效
- 通过user_objects视图可以查看salesman_contacts的状态，注意所有字母要全部大写
- 此时可以删除salesman_contacts视图

openGauss之视图与物化视图

- 官网:

- 视图

- <https://www.opengauss.org/zh/docs/3.1.0/docs/BriefTutorial/%E8%A7%86%E5%9B%BE.html>

- 物化视图

- <https://www.opengauss.org/zh/docs/3.1.0/docs/BriefTutorial/%E7%89%A9%E5%8C%96%E8%A7%86%E5%9B%BE.html>

- 墨天轮:

- <https://www.modb.pro/db/208337>

- <https://www.modb.pro/db/41233>

视图

- 定义视图
- 删除视图
- 查询视图
- **更新视图**
- 视图的作用

更新视图

[例3.95] 将信息系学生视图IS_Student中学号'201215122'的学生姓名改为'刘辰'.

```
UPDATE  IS_Student  
SET    Sname='刘辰'  
WHERE  Sno='201215122';
```



```
UPDATE  Student  
SET    Sname='刘辰'  
WHERE  Sno='201215122' AND Sdept='IS';
```

[例3.96] 向信息系学生视图IS_S中插入一个新的学生记录, 其中姓名为“赵新”, 学号为“201215129”, 年龄为20岁。

```
INSERT  
INTO  IS_Student  
VALUES ('201215129', '赵新', 20);
```



```
INSERT  
INTO  Student(Sno, Sname, Sage, Sdept)  
VALUES ('201215129', '赵新', 20, 'IS');
```

[例3.97] 删除信息系学生视图IS_Student中学号为'201215129'的记录。

```
DELETE  
FROM IS_Student  
WHERE Sno='201215129';
```



```
DELETE  
FROM Student  
WHERE Sno='201215129' AND Sdept='IS';
```

■ 更新视图的限制

- 例3.89定义的视图S_G为不可更新视图

```
UPDATE S_G  
SET   Gavg=90  
WHERE Sno='201215121';
```



这个对视图的更新无法转换成对基本表SC的更新

- 允许对行列子集视图进行更新
- 对其他类型视图的更新不同系统有不同限制
- DB2对视图更新的限制：
 - 若视图是由两个以上基本表导出的，则此视图不允许更新。
 - 若视图的字段来自字段表达式或常数，则不允许对此视图执行INSERT和UPDATE操作，但允许执行DELETE操作。
 - 若视图的字段来自聚集函数，则此视图不允许更新。
 - 若视图定义中含有GROUP BY子句，则此视图不允许更新。
 - 若视图定义中含有DISTINCT短语，则此视图不允许更新。
 - 若视图定义中有嵌套查询，并且内层查询的FROM子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。

[示例]: 将SC中成绩在平均成绩之上的元组定义成一个视图.

```
CREATE VIEW GOOD_SC
AS
SELECT Sno, Cno, Grade
FROM SC
WHERE Grade > (SELECT AVG(Grade)
                FROM SC);
```

- 一个不允许更新的视图上定义的视图也不允许更新

openGauss不支持基于视图的更新!

视图

- 定义视图
- 删除视图
- 查询视图
- 更新视图
- **视图的作用**

视图的作用

- 视图能够简化用户的操作
- 视图使用户能以多种角度看待同一数据
- 视图对重构数据库提供了一定程度的逻辑独立性
- 视图能够对机密数据提供安全保护
- 适当的利用视图可以更清晰的表达查询

- 视图能够简化用户的操作

- 当视图中数据不是直接来自基本表时，定义视图能够简化用户的操作
 - 基于多张表连接形成的视图
 - 基于复杂嵌套查询的视图
 - 含导出属性的视图

- 视图使用户能以多种角度看待同一数据

- 视图机制能使不同用户以不同方式看待同一数据，适应数据库共享的需要

- 视图对重构数据库提供了一定程度的逻辑独立性。
 - 数据库重构
 - 视图只能在一定程度上提供数据的逻辑独立性
 - 由于对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基本表结构的改变而改变。
- 视图能够对机密数据提供安全保护。
 - 对不同用户定义不同视图，使每个用户只能看到他有权看到的数据

[示例]: 学生关系Student(Sno, Sname, Ssex, Sage, Sdept)

- “垂直”地分成两个基本表:
- **SX (Sno, Sname, Sage), SY (Sno, Ssex, Sdept)**
- 通过建立一个视图Student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。

```
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;
```

- 适当的利用视图可以**更清晰的表达查询**。
 - 经常需要执行这样的查询 “对每个同学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;
```



```
SELECT SC.Sno,Cno  
FROM SC,VMGRADE  
WHERE SC.Sno=VMGRADE.Sno AND  
SC.Grade=VMGRADE.Mgrade;
```

本章小结

- SQL可以分为数据定义、数据查询、数据更新、数据控制四大部分。
- SQL是关系数据库语言的工业标准。大部分数据库管理系统产品都能支持SQL 92,但是许多数据库系统只支持SQL 99、SQL2008和SQL2011的部分特征，至今尚没有一个数据库系统能够完全支持SQL99以上的标准。