数据库系统原理与设计

（第3版）

数据库系统原理与设计

（第 3 版）

第7章　SQL数据定义、更新及数据库编程

目　录

SQL数据定义语言

SQL数据更新语言

视　　图

T-SQL语言简介

游　　标

存储过程

触 发 器

7.1　SQL数据定义语言

- 数据库中的关系集合必须由SQL的**数据定义语言**DDL(data definition language)来定义，包括：
  - 数据库、关系模式、每个属性的值域、完整性约束、每个关系的索引集合和关系的物理存储结构等。
- SQL数据定义语言DDL包括：
  - 数据库的定义：创建、修改和删除；
  - 基本表的定义：创建、修改和删除；
  - 视图的定义：创建和删除；
  - 索引的定义：创建和删除。

7.1　SQL数据定义语言

表 7-1　SQL 数据定义

操作对象	创建	修改	删除
数据库	CREATE DATABASE	ALTER DATABASE	DROP DATABASE
基本表	CREATE TABLE	ALTER TABLE	DROP TABLE
视图	CREATE VIEW		DROP VIEW
索引	CREATE INDEX		DROP INDEX

- SQL数据定义语言DDL包括：
  - 数据库的定义：创建、修改和删除；
  - 基本表的定义：创建、修改和删除；
  - 视图的定义：创建、修改和删除；
  - 索引的定义：创建和删除。
- 这些对象的创建、修改和删除方式如表7-1所示：

7.1　SQL数据定义语言

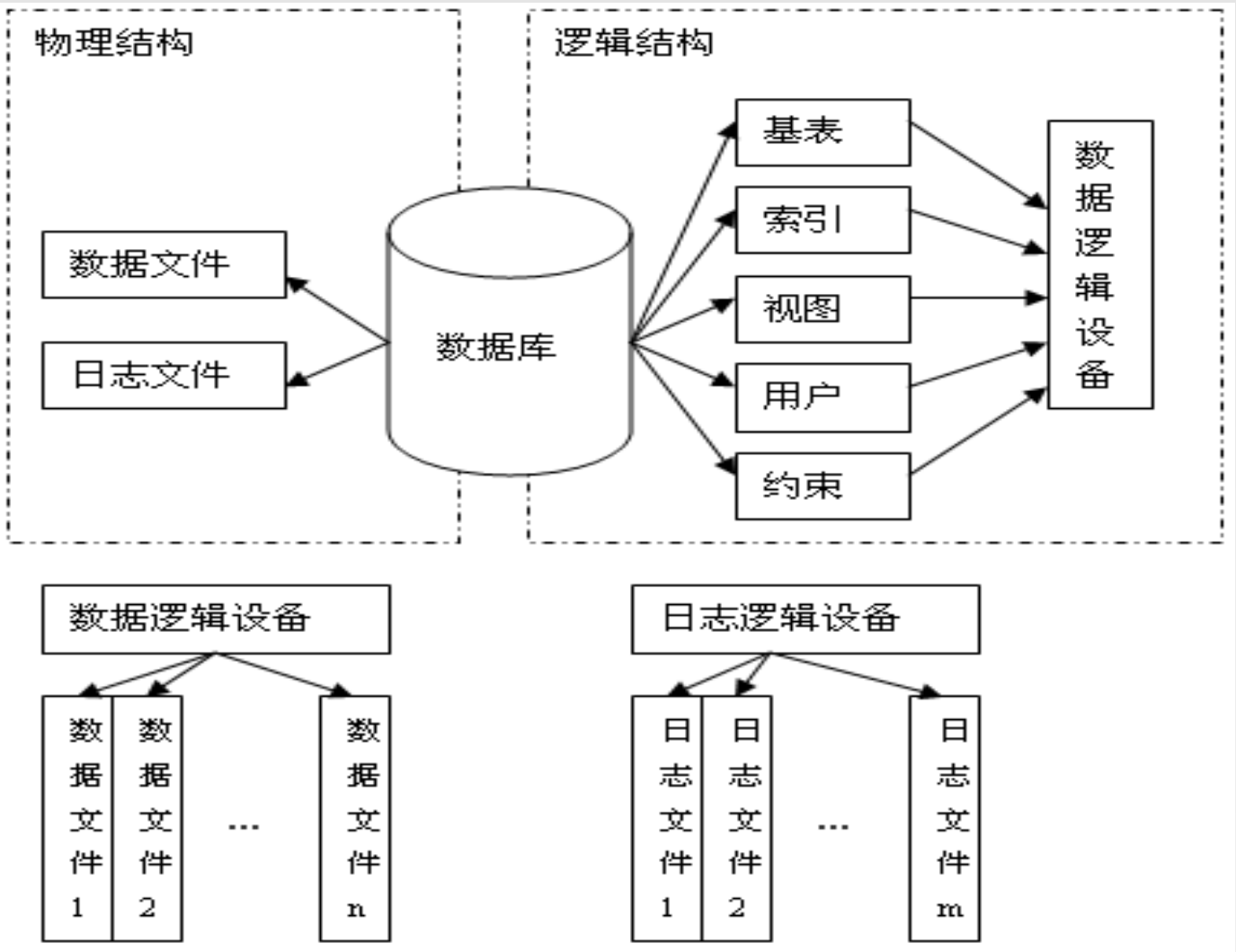
- 7.1.1 数据库的定义
- 7.1.2 基本表的定义
- 7.1.3 索引的定义

7.1.1　数据库的定义

- 数据库保存了企业所有的数据，以及相关的一些控制信息，如安全性和完整性约束、关系的存储路径等。
- 数据库包含了基本表、视图、索引以及约束等对象，在定义这些对象之前，必须首先定义数据库，然后在数据库中定义所有的对象。

7.1.1　数据库的定义

- 数据库保存
  信息，如安
- 数据库包含
  在定义这些
  数据库中定



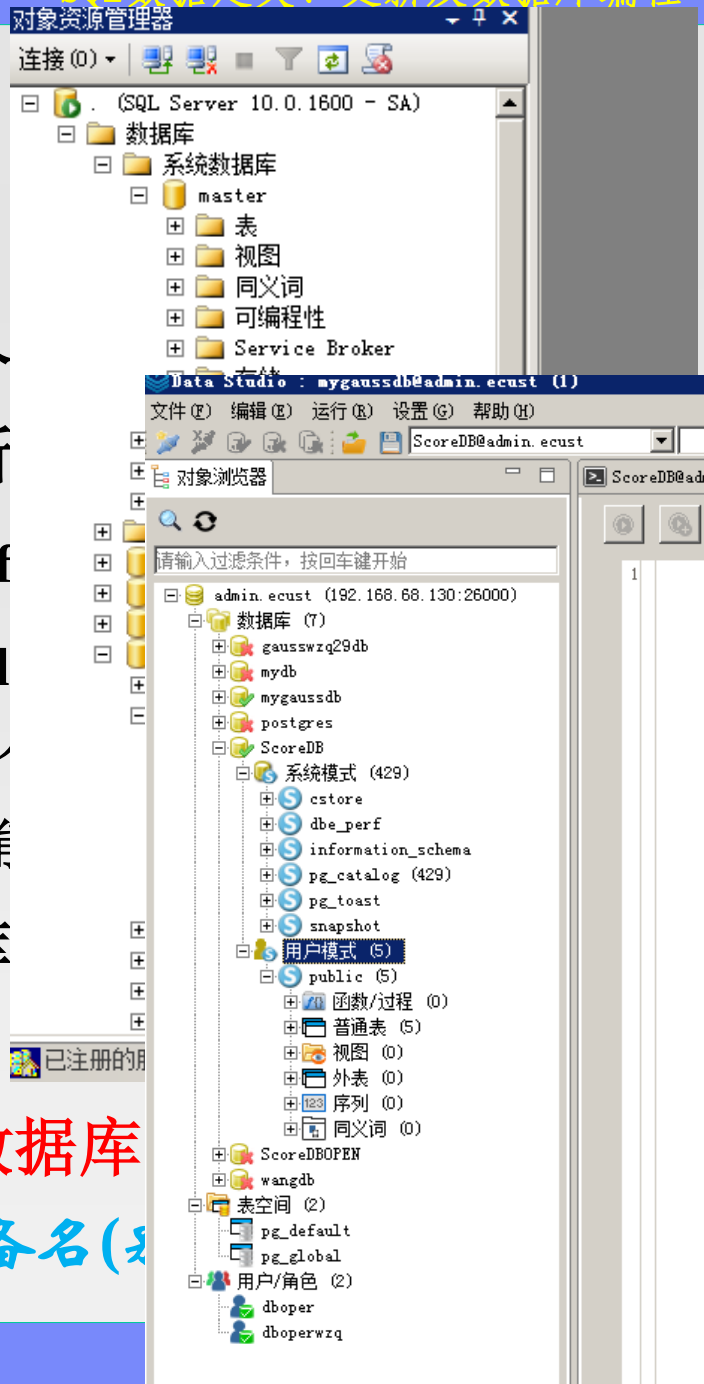
- 数据库与其对象之间的关系如图7-1所示：

7.1.1　数据库的定义

- 数据库作为一个整体存放在外存的物理文件中。
- 物理文件(即磁盘文件)有两种：
  - 一是数据文件，存放数据库中的对象数据；
  - 二是日志文件，存放用于恢复数据库的企业冗余数据。
- 物理文件可以是多个，可以将一个或若干个物理文件设置为一个逻辑设备。
- 数据库可以有多个逻辑设备，必须在定义数据库时进行定义。
- 数据库的对象存放在逻辑设备上，由逻辑设备与物理文件进行联系，从而实现数据库的逻辑模式与存储模式的独立。

7.1.1　数据库的定义：

- 数据库的**创建**
  - 一个数据库创建在物理介质的一个
    分配了将要被数据库和事务日志所
    - 存储数据的文件叫做**数据文件**(data f
    - 存储日志的文件叫做**日志文件**(log fil
    - 创建一个新的数据库时，仅创建了一
      创建**对象**(如表、索引、约束等)，才
  - 当创建了一个数据库，与该数据库
    到**数据字典**(即**数据库系统表**)
  - 在创建数据库的时候，必须定义**数据库**
    (含**数据文件**、**日志文件**)的**逻辑设备名**(



7.1.1 数据库的定义：创建数据库

◆创建数据库操作的语法为：

CREATE DATABASE <*databaseName*>

[ON [PRIMARY] { <*filespec*> [, ... *n*] }　-- 定义主逻辑设备的数据文件

[, { FILEGROUP <*filegroupName*> {<*filespec*> [, ... *n*]} [, ... *n*]}]]

-- 定义用户逻辑设备组的数据文件

[LOG ON { <*filespec*> [, ... *n*] }] -- 定义数据库日志逻辑设备的日志文件

其中：

(1) <*databaseName*>：被创建数据库的名字，满足如下要求：

- 长度从1到30，第一个字符必须是字母，或下划线_，或字符@
- 在首字符后的字符可以是字母、数字或者前面规则中提到的字符
- 名称中不能有空格；
- 数据库的大小可以被扩展或者收缩。

7.1.1　数据库的定义：创建数据库

◆创建数据库操作的语法为：

CREATE DATABASE <*databaseName*>

[ON [PRIMARY] { <*filespec*> [, ... *n*] }　-- 定义主逻辑设备的数据文件

[, { FILEGROUP <*filegroupName*> {<*filespec*> [, ... *n*]} [, ... *n*]}]]

-- 定义用户逻辑设备组的数据文件

[LOG ON { <*filespec*> [, ... *n*] }] -- 定义数据库日志逻辑设备的日志文件

其中：

(2) ON：指定数据库中的数据文件：主数据文件、用户数据文件

- <*filespec*>描述磁盘文件(含数据文件和日志文件)的逻辑文件名(即别名)、物理文件名和磁盘文件初始大小、最大可扩展大小和每次扩展的步长；
- 除了主逻辑设备(PRIMARY)及相关数据文件外，还可以定义用户逻辑设备组(FILEGROUP)及相关数据文件

7.1.1　数据库的定义：创建数据库

◆创建数据库操作的语法为：

CREATE DATABASE <*databaseName*>

[ON [PRIMARY] { <*filespec*> [, ... *n*] }　-- 定义主逻辑设备的数据文件

[, { FILEGROUP <*filegroupName*> {<*filespec*> [, ... *n*]} [, ... *n*]}]]

-- 定义用户逻辑设备组的数据文件

[LOG ON { <*filespec*> [, ... *n*] }]　-- 定义数据库日志逻辑设备的日志文件

其中：

(3) 定义主逻辑设备中的数据文件：[PRIMARY] <*filespec*>

<*filespec*> ::=

([NAME = <*logicalFileName*>,]

FILENAME = '<*osFileName*>' [, SIZE = <*size*>]

[, MAXSIZE = { <*maxSize*> | UNLIMITED }]

[, FILEGROWTH = <*growthIncrement*>])

7.1.1　数据库的定义：创建数据库

◆创建数据库操作的语法为：

CREATE DATABASE <*databaseName*>

[ON [PRIMARY] { <*filespec*> [, ... *n*] }　-- 定义主逻辑设备的数据文件

[, { FILEGROUP <*filegroupName*> {<*filespec*> [, ... *n*]} [, ... *n*]}]]

-- 定义用户逻辑设备组的数据文件

[LOG ON { <*filespec*> [, ... *n*] }]　-- 定义数据库日志逻辑设备的日志文件

其中：

(3) 定义主逻辑设备中的数据文件：[PRIMARY] <*filespec*>

<*filespec*> ::=

([NAME = <*logicalFileName*>,]

FILENAME = '<*osFileName*>' [, SIZE = <*size*>]

(4) 定义用户逻辑设备组中的数据文件：

FILEGROUP <*filegroupName*> <*filespec*>

7.1.1 数据库的定义：创建数据库

(5) **PRIMARY**：描述在主逻辑设备中创建的相关数据文件，所有的数据库系统表存放在主(primary)逻辑设备中，同时也存放没有分配具体逻辑设备的对象

- 在主逻辑设备中第一个数据文件(磁盘文件)称为主(数据)文件，通常包括数据库的逻辑起始位置和系统表
- 对于一个数据库来说，只能有一个主逻辑设备
- 如果主逻辑设备没有指明，则创建数据库时所描述的第一个数据文件将作为主逻辑设备成员

(6) **LOG ON**：描述数据库日志逻辑设备的日志文件 (磁盘文件)

- <*filespec*>描述日志文件。如果没有指定LOG ON，系统将自动创建单个的日志文件

7.1.1　数据库的定义：创建数据库

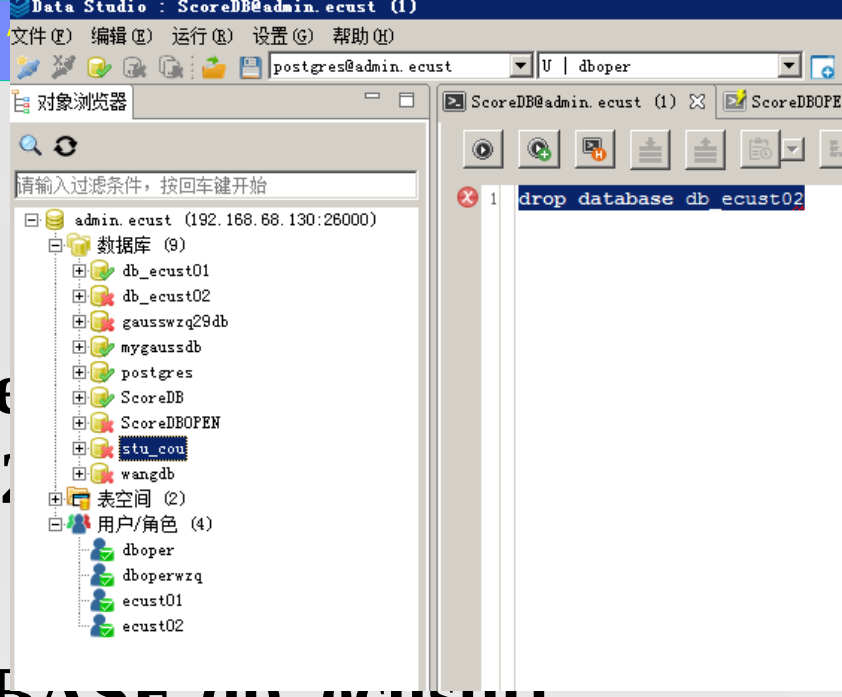
[例7.1] 建立学生成绩数据库ScoreDB

```
CREATE  DATABASE  ScoreDB
ON    -- 定义第一个逻辑设备(默认为主逻辑设备)及其数据文件
    (  NAME=ScoreDB,     -- 数据文件的逻辑文件名(即别名)
       FILENAME='e:\SQLDatabase\ScoreDB.mdf',   -- 物理(磁盘)文件名
       SIZE=2,
       MAXSIZE=10,
       FILEGROWTH=1 )
LOG ON        -- 定义日志逻辑设备及其日志文件
    (  NAME=ScoreLog,     -- 日志文件的逻辑文件名(即别名)
       FILENAME='e:\SQLDatabase\ScoreLog.ldf',   -- 日志(磁盘)文件名
       SIZE=1,
       MAXSIZE=5,
       FILEGROWTH=1 )
```

- openGauss下查看CREATE DATABASE命令的参数可使用下面的命令。
- postgres=#\help CREATE DATABASE
- Command:　CREATE DATABASE
- Description: create a new database
- Syntax:
- CREATE DATABASE database_name
- [[WITH] {[OWNER [=] user_name]|
- [TEMPLATE [=] template]|
- [ENCODING [=] encoding]|
- [LC_COLLATE [=] lc_collate]|
- [LC_CTYPE [=] lc_ctype]|
- [DBCOMPATIBILITY [=] compatibility_type]|
- [TABLESPACE [=] tablespace_name]|
- [CONNECTION LIMIT [=] connlimit]}[...]];

```
postgres=# \help create database
Command:     CREATE DATABASE
Description: create a new database
Syntax:
CREATE DATABASE database_name
    [ [ WITH ] {[ OWNER [=] user_name ]|
           [ TEMPLATE [=] template ]|
           [ ENCODING [=] encoding ]|
           [ LC_COLLATE [=] lc_collate ]|
           [ LC_CTYPE [=] lc_ctype ]|
           [ DBCOMPATIBILITY [=] compatibility_type ]|
           [ TABLESPACE [=] tablespace_name ]|
           [ CONNECTION LIMIT [=] connlimit ]}[...] ];

postgres=#
```



- 1:创建用户
- postgres=# CREATE USER e
  PASSWORD "opengauss@1
- 2：创建数据库
- postgres=# CREATE DATABASE db_ecust01
  OWNER ecust01;
- 3：删除数据库
- drop database db_ecust02;

```
postgres=# drop database db_ecust02;
ERROR:  database "db_ecust02" is being accessed by other users
DETAIL:  There are 3 other sessions using the database.
postgres=# drop database db_ecust02;
DROP DATABASE
postgres=#
```

7.1.1 数据库的定义：修改数据库

■ 修改数据库

- 数据库在运行过程中，可以依据数据量的大小进行修改
- 修改数据库操作的语法为：

```
ALTER DATABASE <databaseName>
{ ADD FILE {<filespec> [, ...n]} [TO FILEGROUP <filegroupName>]
  | ADD LOG FILE {<filespec> [, ... n]}
  | REMOVE FILE <logicalFileName>
  | ADD FILEGROUP <filegroupName>
  | REMOVE FILEGROUP <filegroupName>
  | MODIFY FILE <filespec>
  | MODIFY FILEGROUP <filegroupName> <filegroupProperty>
}
```

7.1.1 数据库的定义：修改数据库

- 数据库在运行过程中，可以依据数据量的大小进行修改
- 修改数据库操作的语法为：

```
ALTER DATABASE <databaseName>
{ ADD FILE <filespec> [,...n] [TO FILEGROUP <filegroupName>]
  | ADD LOG FILE <filespec> [,... n]
  | REMOVE FILE <logicalFileName>
  | ADD FILEGROUP <filegroupName>
  | REMOVE FILEGROUP <filegroupName>
  | MODIFY FILE <filespec>
  | MODIFY FILEGROUP <filegroupName> <filegroupProperty>
}
```

- 其中：
  - ***<databaseName>***：指定被修改的数据库的名字；
  - **ADD FILE** <*filespec*>：指定添加到数据库中的**数据文件**；
  - **TO FILEGROUP** <*filegroupName*>：指定**数据文件**添加到名为<*filegroupName*>的用户**逻辑设备(组)**中；
  - **ADD LOG FILE** <*filespec*>：指定添加到数据库中的**日志文件**；
  - **REMOVE FILE** <*filespec*>：从**数据库系统表**中删除**磁盘文件**(**数据文件**或**日志文件**) <*filespec*>，并物理删除该文件；
  - **ADD FILEGROUP** <*filegroupName*>：指定添加到数据库中的用户**逻辑设备(组)**；
  - **REMOVE FILEGROUP** <*filegroupName*>：从**数据库系统表**中删除指定的用户**逻辑设备(组)**，并删除在该用户**逻辑设备(组)**中的所有**数据文件**；
  - **MODIFY FILE** <*filespec*>：指定要修改的**磁盘文件**(**数据文件**或**日志文件**) <*filespec*>，包含该文件的名称、大小、增长量和最大容量。
- **注意：一次只可以修改其中的一个选项。**

7.1.1 数据库的定义：修改数据库

[例7.3] 修改MyTempDB数据库。

```
ALTER DATABASE MyTempDB
MODIFY FILE ( NAME = TempHisDev1,
              SIZE = 20MB )
```

- 将逻辑文件名(即别名)为TempHisDev1的磁盘文件的初始大小修改为20M

7.1.1 数据库的定义：删除数据库

■ 删除数据库

- 删除数据库时，系统会同时从系统的**数据字典**中将该数据库的描述一起删除
  - ➢ 有的数据库系统会自动删除与数据库相关联的物理文件
- 删除数据库操作的语法为：

  DROP DATABASE *<databaseName>*

数据库定义小结

- 如何创建、删除和修改数据库？
- (数据库，…)都是ADD，ALTER，DROP等
- 两种方式？可视化的，和SQL语言的。

表 3-2　SQL 数据定义

操作对象	创建	修改	删除
数据库	CREATE DATABASE	ALTER DATABASE	DROP DATABASE
表	CREATE TABLE	ALTER TABLE	DROP TABLE
视图	CREATE VIEW	ALTER VIEW	DROP VIEW
索引	CREATE INDEX		DROP INDEX

7.1.2　基本表的定义

- 创建数据库后，就可在数据库中建立基本表。通过将基本表与逻辑设备相关联，使得一个基本表可以放在一个数据文件(磁盘文件)上，也可以放在多个数据文件上。

- SQL中的基本数据类型是：
  - 整型：int (4B)，smallint (2B)，tinyint (1B)；
  - 实型：float，real (4B)，decimal(p, n)，numeric(p, n)；
  - 字符型：char(n)，varchar(n)，text；
  - 2进制型：binary(n)，varbinary(n)，image；
  - 逻辑型：bit，只能取0和1，不允许为空；
  - 货币型：money (8B, 4位小数)，small money (4B, 2位小数)；
  - 时间型：datetime (4B, 从1753.1.1开始)，
    smalldatetime (4B, 从1900.1.1开始)
  - 其中：image为存储图象的数据类型，text存放大文本数据

7.1.2　基本表的定义：创建基本表

■ 创建基本表

- 当创建了一个基本表，与该基本表相关的描述信息会存入到数据字典(即数据库系统表)中。
- 创建基本表操作的语法为：

```
CREATE  TABLE  <tableName>
   (  <columnName1> <dataType>
          [DEFAULT <defaultValue>] [null | NOT null],
     [ <columnName2> <dataType>
          [DEFAULT <defaultValue>] [null | NOT null],  … ]
     [ [CONSTRAINT <constraintName1>] {UNIQUE | PRIMARY  KEY}
          (<columnName> [, <columnName>…]) [ON <filegroupName>], ... ]
     [ [CONSTRAINT <constraintName2>]
          FOREIGN  KEY (<columnName1> [, <columnName2>…] )
          REFERENCE [<dbName>.owner.]<refTable>
               (<refColumn1> [, <refColumn2>… ]) [ON <filegroupName>], ... ]
   ) [ON  <filegroupName>]
```

7.1.2 基本表的定义

● 创建基本表操作的语法为：

```
CREATE TABLE <tableName>
  ( <columnName1> <dataType> [DEFAULT <defaultValue>] [null | NOT null],
    <columnName2> <dataType> [DEFAULT <defaultValue>] [null | NOT null],
    ……
    [CONSTRAINT <constraintName1> {UNIQUE | PRIMARY KEY}
          (<columnName> [, <columnName>…] [ON <filegroupName>] ) ],
    [CONSTRAINT <constraintName2>
          FOREIGN KEY (<columnName1> [, <columnName2>…] ) ],
          [ REFERENCE [<dbName>.owner.]<refTable>
              (<refColumn1> [, <refColumn2>… ] ) ], … ) ON <filegroupName>
  )
```

● 其中：

- <*tableName*>：基本表的名称，最多可包含 128 个字符；
- <*columnName*>：基本表中的列名，在一个基本表内唯一；
- <*dataType*>：指定列的数据类型；
- DEFAULT <*defaultValue*>：为列设置缺省值，属于可选项；
- null | NOT null：为列设置是否允许为空值，属于可选项；
- <*constraintName*>：定义约束的名字，属于可选项；
- UNIQUE：指定某些列的取值必须唯一；
- PRIMARY KEY：建立主码；
- FOREIGN KEY：建立外码；
- ON <*filegroupName*>：将数据库对象放在指定的逻辑设备(组)上
  - ✓ 该逻辑设备(组)必须是在创建数据库时定义的，或者使用数据库的修改命令已加入到数据库中的
  - ✓ 缺省该项时自动将对象建立在主逻辑设备上

7.1.2 基本表的定义：创建基本表

- **建议**：最好不要将**用户对象**建立在**主逻辑设备**上，因为**主逻辑设备**存放了**数据字典**(即**数据库系统表**)。

[例7.4] 建立学生成绩管理数据库中的5张基本表。

```
CREATE TABLE Course (          -- 创建课程表Course
    courseNo      char(3)                       NOT NULL,    --课程号
    courseName    varchar(30)  UNIQUE           NOT NULL,    --课程名
    creditHour    numeric(1)   DEFAULT 0        NOT NULL,    --学分
    courseHour    tinyint      DEFAULT 0        NOT NULL,    --课时数
    priorCourse   char(3)                       NULL,        --先修课程
    /* 建立命名的主码约束和匿名的外码约束 */
    CONSTRAINT CoursePK PRIMARY KEY (courseNo),
    FOREIGN KEY (priorCourse) REFERENCES Course(courseNo)
)          -- 外码约束是匿名的
```

7.1.2 基本表的定义：创建基本表

```
CREATE TABLE Class (                    -- 创建班级表Class
    classNo      char(6)                    NOT NULL,     --班级号
    className    varchar(30) UNIQUE         NOT NULL,     --班级名
    institute    varchar(30)                NOT NULL,     --所属学院
    grade        smallint    DEFAULT 0      NOT NULL,     --年级
    classNum     tinyint                    NULL,         --班级人数
    CONSTRAINT ClassPK PRIMARY KEY (classNo)
)

CREATE TABLE Term (                     -- 创建学期表Term
    termNo       char(3)                    NOT NULL,     --学期号
    termName     varchar(30)                NOT NULL,     --学期描述
    remarks      varchar(10)                NULL,         --备注
    CONSTRAINT TermPK PRIMARY KEY (termNo)
)
```

7.1.2 基本表的定义：创建基本表

```
CREATE TABLE Student (                    -- 创建学生表Student
    studentNo       char(7)                          NOT NULL
        CHECK (studentNo LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),   --学号
    studentName     varchar(20)                      NOT NULL,    --姓名
    sex             char(2)                          NULL,        --性别
    birthday        datetime                         NULL,        --出生日期
    native          varchar(20)                      NULL,        --籍贯
    nation          varchar(30)  DEFAULT '汉族'      NULL,        --民族
    classNo         char(6)                          NULL,        --所属班级
    CONSTRAINT StudentPK PRIMARY KEY (studentNo),
    CONSTRAINT StudentFK FOREIGN KEY (classNo)
                         REFERENCES Class(classNo)
)
```

7.1.2 基本表的定义：创建基本表

```
CREATE TABLE Score (                    -- 创建成绩表Score
    studentNo  char(7)                      NOT NULL ,    --学号
    courseNo   char(3)                      NOT NULL ,    --课程号
    termNo     char(3)                      NOT NULL,     --学期号
    score      numeric(5, 1)  DEFAULT 0     NOT NULL ,    --成绩
    /* 建立由3个属性构成的命名的主码约束 */
    CONSTRAINT ScorePK
        PRIMARY KEY (studentNo, courseNo, termNo),
    /* 建立三个命名的外码约束 */
    CONSTRAINT ScoreFK1 FOREIGN KEY (studentNo)
                        REFERENCES Student(studentNo),
    CONSTRAINT ScoreFK2 FOREIGN KEY (courseNo)
                        REFERENCES Course(courseNo),
    CONSTRAINT ScoreFK3 FOREIGN KEY (termNo)
                        REFERENCES Term(termNo)
)
```

7.1.2 基本表的定义：创建基本表

- 上述5张基本表的创建都缺省了ON <*filegroup_name*>，建立的对象存放在主逻辑设备上。

[例7.5] 在MyTempDB数据库中建立TempTable表，存放在用户逻辑设备(组) TempBakDev上

```
CREATE  TABLE  TempTable (
    xno      char(3)        NOT  NULL,
    xname  varchar(2)    NOT  NULL,
    -- 匿名定义主码约束，由系统对主码约束进行命名
    PRIMARY  KEY (xno)
) ON  TempBakDev   --指定所创建的基本表存放在给定逻辑设备上
```

openGauss讲解

```
CREATE  TABLE class (
        classno character(6) NOT NULL,
        classname character varying(64) NOT NULL,
        institute character varying(32) NOT NULL,
        grade smallint DEFAULT 0::smallint NOT NULL,
        classnum smallint
);
ALTER TABLE class ADD CONSTRAINT class_classname_key
UNIQUE (classname);
ALTER TABLE class ADD CONSTRAINT "ClassPK" PRIMARY KEY
(classno);
```

openGauss讲解

```
CREATE  TABLE course (
        courseno character(3) PRIMARY KEY,
        coursename character varying(30) NOT NULL,
        credithour numeric(1,0) DEFAULT 0::numeric NOT
NULL,
        coursehour smallint DEFAULT 0::smallint NOT NULL,
        priorcourse character(3),
   CONSTRAINT "FK_PRI" FOREIGN KEY (priorcourse)
REFERENCES course(courseno)
);
```

openGauss讲解

```
CREATE  TABLE student (
studentno character(7) PRIMARY KEY,
studentname character varying(24) NOT NULL,
sex character(8),
birthday timestamp(0) without time zone,
native character varying(20),
nation character varying(20),
classno character(6),
    CONSTRAINT "FK-CN0" FOREIGN KEY (classno) REFERENCES
class(classno)
)

CREATE  TABLE term (
termno character(3)  PRIMARY KEY ,
termname character varying(32) NOT NULL,
remarks character varying(16)
)
WITH (orientation=row, compression=no);
ALTER TABLE term ADD CONSTRAINT term_pkey PRIMARY KEY
(termno);
```

openGauss讲解

```
CREATE  TABLE score (
studentno character(7) ,
courseno character(3) ,
termno character(3) ,
score real,
    CONSTRAINT "FK_T" FOREIGN KEY (termno) REFERENCES
term(termno),
    CONSTRAINT "FK_C" FOREIGN KEY (courseno) REFERENCES
course(courseno),
    CONSTRAINT "FK_S" FOREIGN KEY (studentno) REFERENCES
student(studentno)
);
ALTER TABLE score ADD CONSTRAINT "FKSCORE" PRIMARY KEY
(studentno, courseno, termno);
```

openGauss讲解

```
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('CS1501','计算机科学与技术15-01班','信息管理学院',2015,null);
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('CS1502','计算机科学与技术15-02班','信息管理学院',2015,null);
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('IS1501','信息管理与信息系统15-01班','信息管理学院',2015,null);
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('IS1601','信息管理与信息系统16-01班','信息管理学院',2016,null);
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('CP1601','注册会计16_01班','会计学院',2016,null);
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('CP1602','注册会计16_02班','会计学院',2016,null);
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('CP1603','注册会计16_03班','会计学院',2016,null);
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('ER1501','金融管理15-01班','金融学院',2015,null);
INSERT INTO public.class (classno,classname,institute,grade,classnum)
 VALUES ('CS1601','计算机科学与技术16-01班','信息管理学院',2016,null);
```

其他按照次序一次输入

openGauss讲解

可以修改所建表的字段属性数量和类型等

```
alter table c1
add  Sentrance  date;

alter table C1
alter column  Sentrance type SMALLINT;
```

7.1.2 基本表的定义：修改基本表

■ 修改基本表

- 通过ALTER TABLE命令修改基本表的结构，如扩充列等.
- 修改基本表操作的语法为(<*tableName*>为待修改表的名称)：
  - 增加列（新增一列的值为空值）：

```
ALTER TABLE <tableName>
    ADD <columnName> <dataType>
```

  - 增加约束：

```
ALTER TABLE <tableName>
    ADD CONSTRAINT <constraintName>
```

  - 删除约束：

```
ALTER TABLE <tableName>
    DROP <constraintName>
```

  - 修改列的数据类型：

```
ALTER TABLE <tableName>
    ALTER COLUMN <columnName> <newDataType>
```

7.1.2 基本表的定义：修改基本表

[例7.6] 在MyTempDB数据库中为TempTable表增加一列。

```
ALTER TABLE TempTable
    ADD xsex int DEFAULT 0
```

[例7.7] 在MyTempDB数据库中为TempTable表的xname 列修改数据类型。

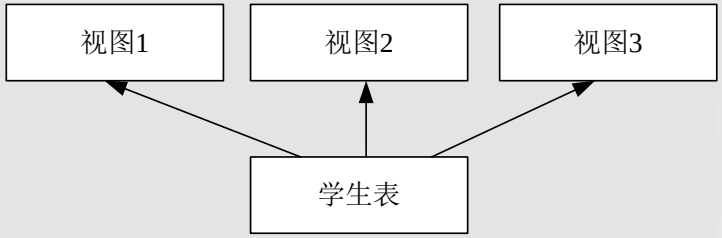
```
ALTER TABLE TempTable
    ALTER COLUMN xname char(10)
```

[例7.8] 在MyTempDB数据库中为TempTable表的xname列增加唯一约束

```
ALTER TABLE TempTable
    ADD CONSTRAINT UniqueXname UNIQUE (xname)
```

注意： 基本表在修改过程中，不可以删除列，一次仅执行一种操作。

7.1.2 基本表的定义：删除基本表



■ 删除基本表

- 删除基本表操作的语法为：

  DROP TABLE <*tableName*> [RESTRICT | CASCADE]

  - <*tableName*>为被删除的基本表的名称
  - 若选择**RESTRICT**，则该基本表的删除有限制条件
    - ✓ 即该基本表不能有视图、触发器以及被其它表所引用(如检查约束CHECK，外码约束FOREIGN KEY)，该项为缺省项。
  - 若选择**CASCADE**，则该基本表的删除没有限制条件
    - ✓ 在删除基本表的同时，也删除建立在该基本表上的所有索引、完整性规则、触发器和视图等。

- 删除基本表时，系统会同时从**数据字典(数据库系统表)**中将该基本表的**描述信息**一起删除。

7.1.2 基本表的定义：删除基本表

[例7.9] 删除TempTable表

```
DROP TABLE TempTable
```

- 注意：SQL Server不支持 [RESTRICT | CASCADE]选项，

基本表的小结

- 知道如何建立基本表，通过命令，或者可视化工具等
- 如何修改基本表，基本表不能删除列等

7.1.3　索引的定义

- 索引是加快数据检索的一种工具
  - 一个基本表可以建立多个索引，从不同角度加快查询速度；
  - 如果索引建立得较多，会给数据维护带来较大的系统开销。
- 索引是由<搜索码值，指针>的记录构成
  - 索引中的记录(称为索引项)按照搜索码值的顺序进行排列，但不改变基本表中记录的物理顺序；
  - 索引和基本表分别存储。

7.1.3　索引的定义

institute	指针
会计学院	
会计学院	
会计学院	
金融学院	
信息管理学院	
信息管理学院	
信息管理学院	
信息管理学院	
信息管理学院	

classNo	className	institute	grade	classNum
CP1601	注册会计16_01班	会计学院	2016	NULL
CP1602	注册会计16_02班	会计学院	2016	NULL
CP1603	注册会计16_03班	会计学院	2016	NULL
CS1501	计算机科学与技术15-01班	信息管理学院	2015	NULL
CS1502	计算机科学与技术15-02班	信息管理学院	2015	NULL
CS1601	计算机科学与技术16-01班	信息管理学院	2016	NULL
ER1501	金融管理15-01班	金融学院	2015	NULL
IS1501	信息管理与信息系统15-01班	信息管理学院	2015	NULL
IS1601	信息管理与信息系统16-01班	信息管理学院	2016	NULL

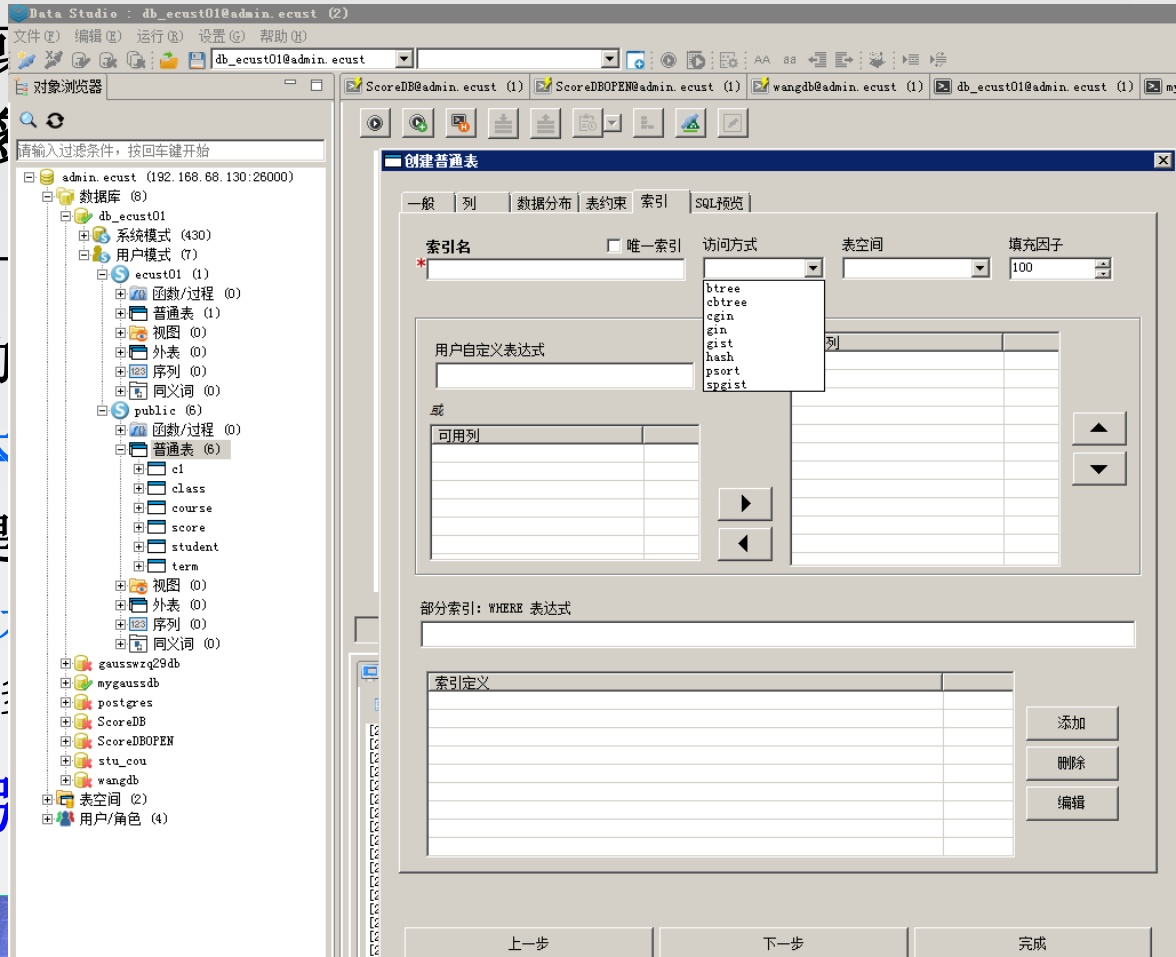
图 7-2　索引与基本表的关系

- 如在班级表Class中按所属学院建立索引InstituteIdx，它与Class表之间的关系可以用图7-2来表示：

什么是位图索引
位图索引是一种使用位图的特殊数据库索引。
主要针对大量相同值的列而创建(例如：类别，操作员，部门ID,库房ID等),
索引块的一个索引行中存储键值和起止Rowid,以及这些键值的位置编码,

义

- 数据库的索引一般按照B^+树结构来组织，但也有Hash索引和位图索引等。
- 索引的类型有：聚集可
  一个基本表可以建立多
- 每个基本表仅能建立一
  - 聚集索引按搜索码值的
  - 即索引的顺序就是基本
  - 聚集索引可以极大地提
    - 建立了聚集索引的基
      数据仓库中使用得较
- 索引创建后，与该索引



7.1.3 索引的定义：建立索引

■ 建立索引操作的语法为：

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX <indexName>
ON <tableName> ( <columnName1> [ASC | DESC]
                [, <columnName2> [ASC | DESC] … ] )
[ON <filegroupName>]
```

其中：

- UNIQUE：建立唯一索引(要求索引属性或属性组上的取值必须唯一)；
- CLUSTERED | NONCLUSTERED：表示建立聚集或非聚集索引，默认为非聚集索引；
- <indexName>：索引的名称，索引是数据库中的对象，因此索引名在一个数据库中必须唯一；
- <tableName> (<columnName1> [ASC | DESC] [, <columnName2> [ASC | DESC] …])：指出为哪个表的哪些属性建立索引
  - ✓ [ASC | DESC]为按升序还是降序建立索引，默认为升序。

7.1.3 索引的定义：建立索引

- ON *<filegroupName>*：指定索引存放在哪个逻辑设备上，该逻辑设备必须是创建数据库时定义的，或通过修改已加入到数据库中的。
  - ✓ 缺省该项时，自动将该索引建立在主逻辑设备上。

[例7.10] 对班级表Class按所属学院建立非聚集索引InstituteIdx.

```
CREATE NONCLUSTERED INDEX InstituteIdx
ON Class(institute)
```

[例7.11] 在学生表Student中，首先按班级编号的升序，然后按出生日期的降序建立一个非聚集索引ClassBirthIdx。

```
CREATE INDEX ClassBirthIdx
ON Student(classNo, birthday DESC)
```

7.1.3 索引的定义：删除索引

- 删除索引
  - 索引一旦建立，用户不需要管理它，由系统自动维护；
  - 可删除那些不经常使用的索引。
- 删除索引操作的语法为：

  DROP INDEX *<indexName>* **ON** *<tableName>*

  - 索引被删除时，系统会同时从**数据字典(数据库系统表)**中将该索引的**描述信息**一起删除。

[例7.12] 删除班级表Class中建立的InstituteIdx索引。

DROP INDEX InstituteIdx ON Class

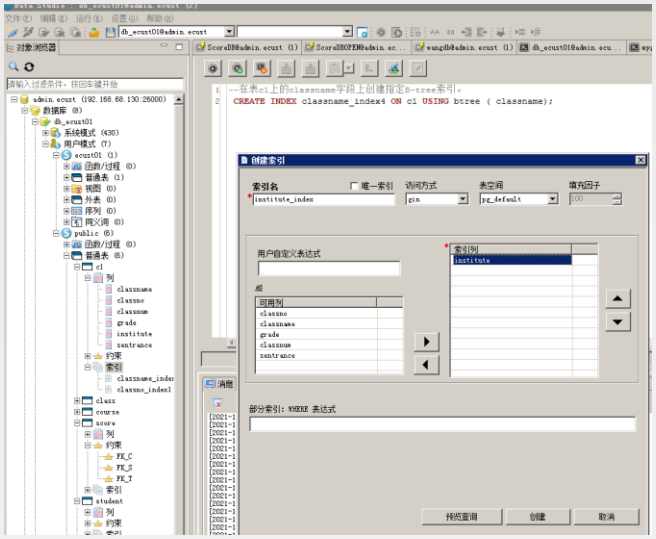
openGauss讲解

在表上创建索引。CREATE [UNIQUE] INDEX [[schemaname.]index_name]
ON table_name [USING method]
　({ { column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS { FIRST | LAST }] }[, ...])
　[WITH ({storage_parameter = value} [, ...])]
　[TABLESPACE tablespace_name]
　[WHERE predicate];

openGauss讲解

```
--在表c1上的classno字段上创建普通索引。
CREATE UNIQUE INDEX classno_index1 ON c1( classno );
```

```
--在表c1上的classname字段上创建指定B-tree索引。
 CREATE INDEX classname_index4 ON c1 USING btree (
classname);
```



删除索引：1）使用窗体界面删除；2）使用命令行删除。

--在表c1上的classname字段上创建指定B-tree索引。

DROP INDEX classname_index4

- 索引
- 1 如何建立索引？B+，pg系列, hash，位图索引
- 2 索引分为聚集或非聚集
- 3 如何删除

目　录

SQL数据定义语言

SQL数据更新语言

视　　图

T-SQL语言简介

游　　标

存储过程

触 发 器

7.2　SQL数据更新语言

- 结构化查询语言SQL包括：数据定义语言DDL、数据操纵语言DML、数据控制语言DCL和其它。
- 数据操纵语言DML(data manipuplation language)主要用于对数据库的数据进行检索(查询)和更新。
- SQL数据更新语句包括三条：插入INSERT、删除DELETE和修改UPDATE。
- 7.2.1 插入数据
- 7.2.2 删除数据
- 7.2.3 修改数据

7.2.1　插入数据

- 插入方式有两种：一是插入一个元组(即记录)，二是插入子查询的结果。后者是一次插入多个元组。
- 插入一个元组：

INSERT INTO <*tableName*> [(<*columnName1*> [, <*columnName2*> ...])]
VALUES (<*value1*> [, <*value2*> ...])

  - 功能：将新元组插入到指定的基本表中。
  - <*tableName*>：要插入元组的基本表的名称；
  - <*columnName1*> [, <*columnName2*> ...]：指明被插入的元组按<*columnName1*>, <*columnName2*>, ...指定的属性名称和顺序插入元组数据到基本表<*tableName*>中去。
    - 该项可以省略。若省略，表示必须按照<*tableName*>表的属性个数和属性顺序插入新元组。

7.2.1　插入数据

- <*value1*> [, <*value2*> ...]：指明被插入元组的具体属性值。
  - 值的个数和顺序必须与 <*columnName1*> [, <*columnName2*> ...]相对应;
  - 对于<*tableName*> 表中的属性没有在<*columnName1*> [, <*columnName2*> ...]中出现的属性列，系统自动取空值;
  - 注意：如果不在 <*columnName1*> [, <*columnName2*> ...]中出现的属性被定义为非空值，则该插入语句会报错。

[例7.13] 将一个新学生元组（'0700006', '李相东', '男', '1991-10-21 00:00', '云南', '撒呢族', 'CS0701'）插入到学生表Student中。

```
INSERT INTO Student
VALUES ( '0700006', '李相东', '男', '1991-10-21 00:00', '云南',
         '撒呢族', 'CS0701' )
```

- 本例表名Student后没有指定属性名，表示按照Student表定义的属性列的个数和顺序将新元组插入到Student表中。

7.2.1　插入数据

[例7.14] 将一个新学生元组（姓名：章李立，出生日期：1991-10-12 00:00，学号：1500007）插入到学生表Student中。

```
INSERT  INTO  Student(studentName, birthday, studentNo)
VALUES ( '章李立', '1991-10-12 00:00', '0700007' )
```

- 本例按照指定属性的顺序和属性的个数向学生表Student插入一个新元组，没有列出的属性列自动取空值NULL或默认值；
- 插入新元组时，数据的组织可不按照基本表结构定义的属性个数和顺序进行插入。

7.2.1　插入数据

■ 插入多个元组：

INSERT INTO *<tableName>* [(*<columnName1>* [, *<columnName2>* ...])]
　　<subquery>

其中：

- *<tableName>*：要插入元组的基本表的名称；
- *<columnName1>* [, *<columnName2>* ...]：指明被插入的元组按 *<columnName1>*, *<columnName2>*, ... 指定的属性名称和顺序插入到基本表*<tableName>*中；
  - 该项可以省略，若省略则其查询出来的结果必须与*<tableName>*表结构相同；
- *<subquery>*：由SELECT语句引出的一个子查询。

7.2.1　插入数据

[例7.15] 将少数民族同学的选课信息插入基本表StudentNation中.

- 首先，创建基本表StudentNation：

```
CREATE TABLE StudentNation (
    studentNo  char(7)                              NOT NULL ,    --学号
    courseNo   char(3)                              NOT NULL ,    --课程号
    termNo     char(3)                              NOT NULL,     --学期号
    score      numeric(5, 1)  DEFAULT 0  NOT NULL                 --成绩
        CHECK( score BETWEEN 0.0 AND 100.0),
    CONSTRAINT StudentNationPK
        PRIMARY KEY (studentNo, courseNo, termNo)
)
```

- 然后，执行如下插入语句：

```
INSERT INTO StudentNation
  SELECT *
  FROM Score
  WHERE studentNo IN (
      SELECT studentNo FROM Student WHERE nation<>'汉族' )
```

openGauss讲解

```
CREATE  TABLE class (
                classno character(6) NOT NULL,
                classname character varying(64) NOT NULL,
                institute character varying(32) NOT NULL,
                grade smallint DEFAULT 0::smallint NOT NULL,
                classnum smallint
);
ALTER TABLE class ADD CONSTRAINT class_classname_key UNIQUE (classname);
ALTER TABLE class ADD CONSTRAINT "ClassPK" PRIMARY KEY (classno);
CREATE  TABLE student (
studentno character(7) PRIMARY KEY,
studentname character varying(24) NOT NULL,
sex character(8),
birthday timestamp(0) without time zone,
native character varying(20),
nation character varying(20),
classno character(6),
   CONSTRAINT "FK-CN0" FOREIGN KEY (classno) REFERENCES class(classno)
```

插入多个元组：

```
INSERT INTO StudentNa
        SELECT * FROM STUDENT WHERE
nation<>'汉族'
```

7.2.1　插入数据

[例7.16] 将汉族同学的选课信息插入到StudentNation表中。

```
INSERT  INTO  StudentNation(studentNo, courseNo, termNo)
    SELECT  studentNo, courseNo, termNo
    FROM  Score
    WHERE  studentNo  IN  (
                SELECT  studentNo
                FROM  Student
                WHERE  nation='汉族' )
```

- 该查询仅将汉族同学的学号、课程号和学期号插入到基本表StudentNation中；
- 成绩列自动取0值，而不是空值NULL，这是因为在定义基本表StudentNation时该列设置了默认值为0。

7.2.2　删除数据

■ 删除命令：

DELETE FROM <*tableName*> [WHERE <*predicate*>]

- <*tableName*>：要删除记录的基本表的名称；
- [WHERE <*predicate*>]：指出被删除的元组所满足的条件
  - ➢该项可以省略，若省略则表示删除表中的所有元组；
  - ➢WHERE子句中可以包含子查询。

[例7.17] 删除学号为1600001同学的选课记录。

```
DELETE FROM Score
WHERE studentNo='1600001'
```

7.2.2　删除数据

[例7.18] 删除选修了“高等数学”课程的选课记录。

```
DELETE FROM Score
WHERE courseNo IN (
        SELECT courseNo
        FROM Course
        WHERE courseName='高等数学' )
```

[例7.19] 删除平均分在60到70分之间的同学的选课记录

```
DELETE FROM Score
WHERE studentNo IN (
        SELECT studentNo
        FROM Score
        GROUP BY studentNo
        HAVING avg(score) BETWEEN 60 AND 70 )
```

7.2.3　修改数据

■ 修改数据命令：

UPDATE *<tableName>*
SET *<columnName1>* = *<expr1>* [, *<columnName2>* = *<expr2>* ...]
[**FROM** {*<tableName1>* | *<queryName1>* | *<viewName1>*} [**AS**] [*<aliasName1>*]
[, {*<tableName2>* | *<queryName2>* | *<viewName2>*} [**AS**] [*<aliasName2>*] ...]
[**WHERE** *<predicate>*]

其中：

- *<tableName>*：要进行修改数据的基本表的名称。
- **SET** *<columnName1>* = *<expr1>* [, *<columnName2>* = *<expr2>* ...]：用表达式的值替代属性列的值
  - 一次可以修改元组的多个属性列，之间以逗号分隔。
- [**WHERE** *<predicate>*]：指出被修改的元组所满足的条件
  - 该项可以省略，若省略，表示修改基本表中的所有元组；
  - **WHERE**子句中可以包含子查询。

7.2.3 修改数据

[例7.20] 将王红敏同学在151学期选修的002课程的成绩改为88分。

```
UPDATE Score
SET score=88
WHERE courseNo='002' AND termNo='151'
    AND studentNo IN
        ( SELECT studentNo FROM Student
          WHERE studentName='王红敏' )
```

studentNo	courseNo	termNo	score
1500001	001	151	98.0
1500001	002	151	82.0
1500001	003	161	82.0
1500001	004	151	56.0
1500001	004	161	86.0
1500001	005	152	77.0
1500001	006	152	76.0
1500001	007	152	77.0
1500001	008	161	82.0
1500001	009	162	77.0
1500001	010	151	86.0
1500003	001	151	46.0
1500003	002	151	38.0
1500003	002	152	58.0
1500003	005	151	60.0
1500003	006	161	70.0
1500003	007	152	50.0
1500003	007	162	66.0

也可以写成：

```
UPDATE Score
SET score=88
FROM Score a, Student b
WHERE a.studentNo=b.studentNo AND a. courseNo='002'
    AND a.termNo='151' AND studentName='王红敏'
```

将注册会计16_02班的男同学的成绩都增加5分。

7.2.3 修改数据

[例7.21] 将注册会计16_02班的男同学的成绩都增加5分。

```
UPDATE  Score
SET  score=a.score+5
FROM  Score a, Student b, Class c
WHERE  a.studentNo=b.studentNo  AND  b.classNo=c.classNo
    AND  className='注册会计16_02班'  AND  sex='男'
```

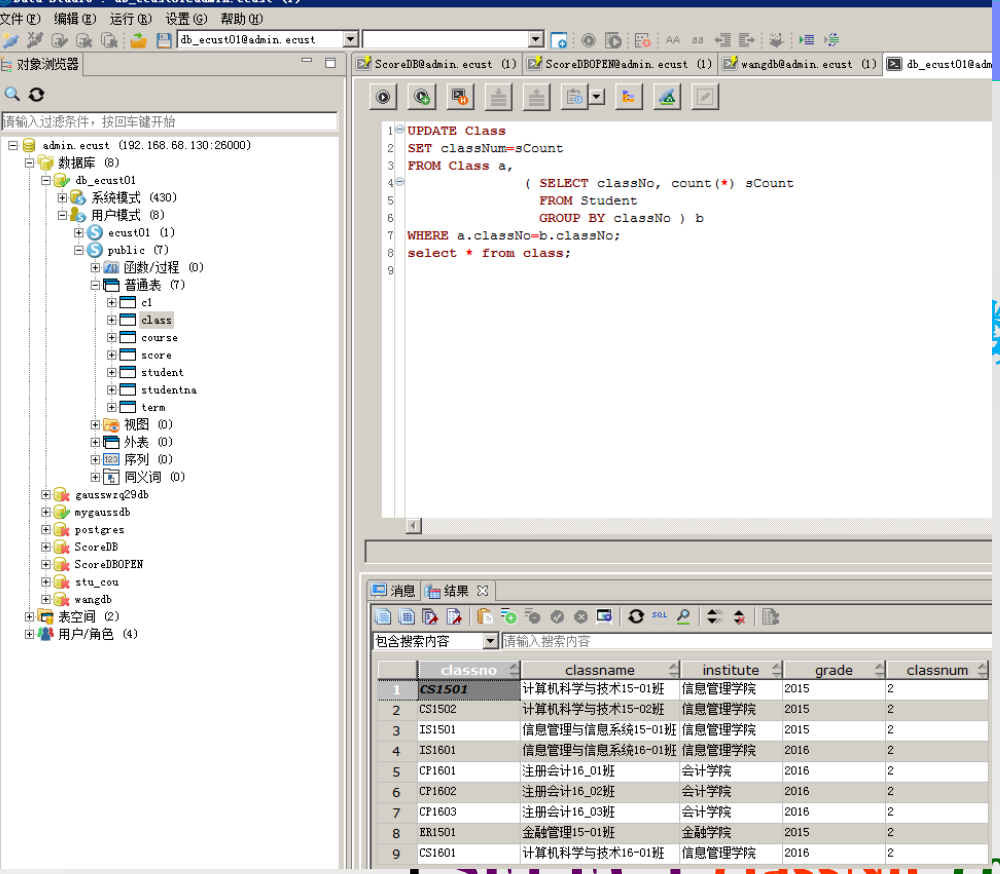
[例7.22] 将学号为1600001同学的出生日期修改为1999年5月6日出生，籍贯修改为福州。

```
UPDATE  Student
SET  birthday='1999-5-6 00:00', native='福州'
WHERE  studentNo='0800001'
```

- 注意：插入、删除和修改操作会破坏数据的完整性，如果违反了完整性约束条件，其操作会失败。

改数据

数填入到班级表的ClassNum



	classno	classname	institute	grade	classnum
1	CS1501	计算机科学与技术15-01班	信息管理学院	2015	2
2	CS1502	计算机科学与技术15-02班	信息管理学院	2015	2
3	IS1501	信息管理与信息系统15-01班	信息管理学院	2015	2
4	IS1601	信息管理与信息系统16-01班	信息管理学院	2016	2
5	CP1601	注册会计16_01班	会计学院	2016	2
6	CP1602	注册会计16_02班	会计学院	2016	2
7	CP1603	注册会计16_03班	会计学院	2016	2
8	ER1501	金融管理15-01班	金融学院	2015	2
9	CS1601	计算机科学与技术16-01班	信息管理学院	2016	2

	studentNo	studentName	sex	birthday	native	nation	classNo
1	1500001	李小勇	男	1998-12-21 00:00:00.000	南昌	汉族	CS1501
2	1500002	刘方晨	女	1998-11-11 00:00:00.000	九江	汉族	IS1501
3	1500003	王红敏	女	1997-10-01 00:00:00.000	上海	汉族	IS1501
4	1500004	张可立	男	1999-05-20 00:00:00.000	南昌	蒙古族	CS1501
5	1500005	王红	男	2000-04-26 00:00:00.000	南昌	蒙古族	CS1502
6	1600001	李勇	男	1998-12-21 00:00:00.000	南昌	汉族	CS1601
7	1600002	刘晨	女	1998-11-11 00:00:00.000	九江	汉族	IS1601
8	1600003	王敏	女	1998-10-01 00:00:00.000	上海	汉族	IS1601
9	1600004	张立	男	1999-05-20 00:00:00.000	南昌	蒙古族	CS1601
10	1600005	王红	男	1999-04-26 00:00:00.000	南昌	蒙古族	CP1602
11	1600006	李志强	男	1999-12-21 00:00:00.000	北京	汉族	CP1602
12	1600007	李立	女	1999-08-21 00:00:00.000	福建	畲族	IS1601
13	1600008	黄小红	女	1999-08-09 00:00:00.000	云南	傣族	CS1601
14	1600009	黄勇	男	1999-11-21 00:00:00.000	九江	汉族	CP1602
15	1600010	李宏冰	女	1998-03-09 00:00:00.000	上海	汉族	CP1602
16	1600011	江宏吕	男	1998-12-20 00:00:00.000	上海	汉族	CP1602
17	1600012	王立红	男	1998-11-18 00:00:00.000	北京	汉族	CS1601
18	1600013	刘小华	女	1999-07-16 00:00:00.000	云南	哈呢族	IS1601
19	1600014	刘宏昊	男	1999-09-16 00:00:00.000	福建	汉族	IS1601

(SELECT classNo, count(*) sCount

FROM Student

GROUP BY classNo) *b*

WHERE *a*.classNo=*b*.classNo

	classNo	className	institute	grade	classNum
1	CP1601	注册会计16_01班	会计学院	2016	NULL
2	CP1602	注册会计16_02班	会计学院	2016	NULL
3	CP1603	注册会计16_03班	会计学院	2016	NULL
4	CS1501	计算机科学与技术15-01班	信息管理学院	2015	NULL
5	CS1502	计算机科学与技术15-02班	信息管理学院	2015	NULL
6	CS1601	计算机科学与技术16-01班	信息管理学院	2016	NULL
7	ER1501	金融管理15-01班	金融学院	2015	NULL
8	IS1501	信息管理与信息系统15-01班	信息管理学院	2015	NULL
9	IS1601	信息管理与信息系统16-01班	信息管理学院	2016	NULL

- 数据更新操作
- 插入，删除，修改
- 插入一条记录多条记录？
- 删除表和表记录？
- 修改？
- 另外特别要注意删除关联表中数据时，看清题意。

目　录

SQL数据定义语言

SQL数据更新语言

视　　图

T-SQL语言简介

游　　标

存储过程

触 发 器

7.3 视图

- 视图是虚表，是从一个或几个基本表(或视图)中导出的表。
- 在数据字典(数据库系统表)中仅存放创建视图的语句，不存放视图对应的数据。
- 当基本表中的数据发生变化时，从视图中查询出的数据也随之改变。
- 视图实现了数据库管理系统三级模式中的外模式。
- 基于视图的操作包括：
  - 查询、删除、受限更新和创建基于该视图的新视图。
- 视图的主要作用是：
  - 简化用户的操作；
  - 使用户能以多种角度看待同一数据库模式；
  - 对重构数据库模式提供了一定程度的逻辑独立性；
  - 能够对数据库中的机密数据提供一定程度的安全保护；
  - 适当的利用视图可以更清晰的表达查询。

7.3　视图

- 7.3.1 创建视图
- 7.3.2 查询视图
- 7.3.3 更新视图
- 7.3.4 删除视图

7.3.1 创建视图

■ 创建视图：

CREATE VIEW *<viewName>* **[(***<columnName1>* **[,** *<columnName2>* **...])]**
AS
　　<subquery>
[WITH CHECK OPTION]

其中：

- *<viewName>*：新建视图的名称，该名称在一个数据库中必须唯一；
- *<columnName1>* **[,** *<columnName2>* **...]**：视图中定义的列名，如果列名省略，列名自动取查询出来的列名，但属于下列3种情况必须写列名：
  - 某个目标列是聚集函数或表达式；
  - 多表连接中有相同的列名；
  - 在视图中为某列取新的名称更合适。
- **AS** *<subquery>*：子查询不允许含有**ORDER BY**子句和**DISTINCT**短语
- **[WITH CHECK OPTION]**：当对视图进行插入、删除和修改操作时，必须满足创建视图中的谓词条件(即子查询*<subquery>*中的条件表达式)

7.3.1 创建视图

- 执行CREATE VIEW语句时只把创建视图的语句存入数据字典中，并不执行其中的SELECT语句

[例7.24] 创建仅包含1999年出生学生的视图StudentView1999。

```
CREATE VIEW StudentView1999
AS
    SELECT *
    FROM Student
    WHERE year(birthday)=1999   -- 创建视图中的谓词条件
```

- 本例省略了视图的列名，自动取查询出来的列名。
- 本例没有使用WITH CHECK OPTION选项，下面的插入语句可以执行：

```
INSERT INTO StudentView1999 VALUES
    ('1500008', '李相东', '男', '1998-10-21 00:00', '云南', '撒呢族', 'CS1501')
```

- 但是, 对视图StudentView1999查询时却不能查询出刚插入的元组, 因为刚插入的学生元组并不满足创建视图中的谓词条件。

7.3.1 创建视图

- 执行CREATE VIEW语句时只把创建视图的语句存入数据字典中，并不执行其中的SELECT语句(openGauss命令下)

[例7.24] 创建仅包含1999年出生学生的视图StudentView1999。

```
CREATE VIEW StudentView1999
AS
    SELECT *
    FROM Student
    WHERE date_part('year', birthday)=1999 -- 创建视图中的谓词条件
```

- 本例省略了视图的列名，自动取查询出来的列名。
- 本例没有使用WITH CHECK OPTION选项，下面的插入语句可以执行：

```
INSERT INTO StudentView1999 VALUES
    ('1500008', '李相东', '男', '1998-10-21 00:00', '云南', '撒呢族', 'CS1501')
```

- 但是, 对视图StudentView1999查询时却不能查询出刚插入的元组, 因为刚插入的学生元组并不满足创建视图中的谓词条件。

7.3.1 创建视图

[例7.25] 创建仅包含1999年出生学生的视图StudentView1999Chk，并要求在对该视图进行更新操作时，进行合法性检查(即保证更新操作要满足创建视图中的谓词条件)。

```
CREATE VIEW StudentView1999Chk
AS
    SELECT *
    FROM Student
    WHERE year(birthday)=1999      -- 创建视图中的谓词条件
WITH CHECK OPTION
```

- 本例建立的视图StudentView1999Chk，其更新操作必须满足：
  - 修改操作：自动加上year(birthday)=1999的条件；
  - 删除操作：自动加上year(birthday)=1999的条件；
  - 插入操作：自动检查是否满足条件year(birthday)=1999，如果不满足，则拒绝该插入操作。

7.3.1 创建视图

- 本例使用了WITH CHECK OPTION选项，下面的插入语句可以执行：

```
INSERT INTO StudentView1999Chk VALUES
('1500008', '李相西', '男', '1999-10-21 00:00', '云南', '撒呢族', 'CS1501')
```

基于连接和聚集函数的视图

- 下面的插入语句不可以执行：

```
INSERT INTO StudentView1999Chk VALUES
('0700009', '李相南', '男', '1998-10-21 00:00', '云南', '撒呢族', 'CS1501')
```

  - 原因是插入的出生日期不满足条件year(birthday)=1999。

- 当视图是基于一个基本表创建的，且保留了主码属性，这样的视图称为行列子集视图。
- 视图可建立在一个基本表上，也可以建立在多个基本表上。

7.3.1 创建视图

[例7.26] 创建一个包含学生学号、姓名、课程名、获得的学分和相应成绩的视图ScoreView。

- 由于成绩必须大于等于60分才获得学分，该视图必须含有该条件。

```
CREATE VIEW ScoreView
AS
    SELECT a.studentNo, studentName, courseName, creditHour, score
    FROM Student a, Course b, Score c
    WHERE a.studentNo=c.studentNo AND b.courseNo=c.courseNo
        AND score>=60    -- 成绩必须大于等于60分才能获得学分
```

- 但是，如果某学生选修某门课程2次且都考试及格，则视图的查询结果中会出现2次。

7.3.1 创建视图

[例7.27] 创建一个包含每门课程的课程编号、课程名称、选课人数和选课平均成绩的视图SourceView。

```
CREATE VIEW SourceView(courseNo, courseName, courseCount, courseAvg)
AS
    SELECT a.courseNo, courseName, count(*), avg(score)
    FROM Course a, Score b
    WHERE a.courseNo=b.courseNo
    GROUP BY a.courseNo, courseName
```

- 本例使用聚合函数，必须为视图的属性命名，可在视图名的后面直接给出列名，也可用下面的语句替代：

```
CREATE VIEW SourceView1
AS
    SELECT a.courseNo, courseName, count(*) courseCount, avg(score) courseAvg
    FROM Course a, Score b
    WHERE a.courseNo=b.courseNo
    GROUP BY a.courseNo, courseName
```

7.3.1 创建视图

■ 视图也可以建立在视图上。

[例7.28] 创建一个包含每门课程的课程编号、课程名称、选课人数和选课平均成绩的视图SourceView2，要求该视图选课人数必须在5人以上。

```
CREATE VIEW SourceView2
AS
    SELECT *
    FROM SourceView
    WHERE courseCount>=5          -- 创建视图中的谓词条件
```

■ 在设计表结构时，为减少数据的冗余存放，往往仅存放最基本的数据：

- 凡是可以由基本数据导出的数据，在基本表中一般不存储；
- 如在学生Student表中没有存放年龄，但可建立一个包含年龄属性的视图，这样的视图称为带表达式的视图。

7.3.1 创建视图

[例7.29] 创建一个包含学生学号、姓名和年龄的视图StudentAgeView。(SQL Sever)

```
CREATE VIEW StudentAgeView AS
    SELECT studentNo, studentName, year(getdate())-year(birthday) age
    FROM Student
```

openGauss

```
CREATE VIEW StudentAgeView AS
      SELECT studentNo, studentName,
date_part('year',CURRENT_TIME)-
date_part('year',birthday) age
      FROM Student
```

7.3.2 查询视图

- 查询是对视图进行的最主要的操作。
- 从用户的角度来看，查询视图与查询基本表的方式是完全一样的。
- 从系统的角度来看，查询视图的过程是：
  - 有效性检查：检查查询中涉及的基本表和视图是否存在？
  - 从**数据字典**中取出创建视图的语句，将创建视图的子查询与用户的查询结合起来，转换成等价的对基本表的查询；
  - 执行改写后的查询。

7.3.2 查询视图

[例7.30] 在视图StudentView1999中查询CS1601班同学的信息。

```
SELECT *
FROM StudentView1999
WHERE classNo='CS1601'
```

对于该查询：

- 系统首先进行有效性检查：判断视图StudentView1999是否存在？
- 如果存在，则从系统的数据字典中取出该创建视图的语句；
- 将创建视图中的子查询与用户的查询结合起来，转换为基于基本表的查询，即将视图StudentView1999的定义转换该查询为：

```
SELECT *
FROM Student
WHERE year(birthday)=1999 AND classNo='CS1601'
```

- 然后系统执行改写后的查询。

7.3.2　查询视图

[例7.31] 在视图SourceView中查询平均成绩在80分以上的课程信息。

```
SELECT *
FROM SourceView
WHERE courseAvg>=80
```

课程编号	课程名称	选课人数	平均成绩
001	大学语文	12	69.750000
002	体育	11	66.272727
003	大学英语	8	81.500000
004	高等数学	11	68.363636
005	C语言程序设计	8	77.375000
006	计算机原理	5	73.800000
007	数据结构	6	74.000000
008	操作系统	4	82.750000
009	数据库系统原理	3	81.666666
010	会计学原理	8	83.500000
011	中级财务会计	4	77.500000

- 视图SourceView是一个基于聚合运算的视图，是经过聚合函数运算的值。
- 由于在WHERE子句中，不允许对聚合函数进行运算，不可能转换为如下的查询：

```
SELECT a.courseNo, courseName, count(*) courseCount, avg(score) courseAvg
FROM Course a, Score b
WHERE a.courseNo=b.courseNo AND courseAvg>=80
GROUP BY a.courseNo, courseName
```

7.3.2 查询视图

- HAVING子句可以对聚合函数直接作用，系统会将该查询转换为如下的形式：

```
SELECT  a.courseNo, courseName, count(*) courseCount, avg(Score) courseAvg
FROM  Course a, Score b
WHERE  a.courseNo=b.courseNo
GROUP  BY  a.courseNo, courseName
HAVING  avg(score)>=80
```

[例7.32] 在视图SourceView和课程表Course中查询课程平均成绩在75分以上的课程编号、课程名称、课程平均成绩和学分。

```
SELECT  a.courseNo, a.courseName, courseAvg, creditHour
FROM  Course a, SourceVIEW b
WHERE  a.courseNo=b.courseNo
      AND  courseAvg>=75
```

7.3.3 视图更新

- 视图更新指通过视图来插入、删除和修改基本表中的数据。
- 视图不实际存放数据，对视图的更新，最终要转换为对基本表的更新。
  - 如果创建视图的语句中包含了表达式，或聚合运算，或消除重复值运算，则不能对视图进行更新操作。
- 对视图进行更新操作，其限制条件比较多
  - 建立视图的作用不是利用视图来更新数据库中的数据，而是简化用户的查询；
  - 达到一定程度的安全性保护；
  - **尽量不要对视图执行更新操作。**

7.3.3 视图更新

[例7.33] 在视图StudentView1999中，将学号为'1600004'同学的姓名修改为'张小立'。

```
UPDATE  StudentView1999
SET  studentName='张小立'
WHERE  studentNo='1600004'
```

对于该操作：

- 系统首先进行有效性检查：判断视图StudentView1999是否存在？
- 如果存在，则从系统的数据字典中取出该创建视图的语句；
- 将创建视图中的子查询与用户查询相结合，转换为对基本表的修改：

```
UPDATE  Student
SET  studentName='张小立'
WHERE  year(birthday)=1999  AND  studentNo='1600004'
```

7.3.3 视图更新

[例7.34] 在视图StudentView1999中将学号为'1600004'同学的出生年份由1999修改为2000。

```
UPDATE  StudentView1999
SET  birthday='2000-05-20 00:00:00.000'
WHERE  studentNo='1600004'
```

- **注意**：在视图StudentView1999Chk中不能将出生年份修改为2000，因为该视图对修改操作进行了检查(即检查修改后的结果是否满足创建视图中的谓词条件)。对视图的插入操作见例7.24和例7.25。

7.3.3　视图更新

[例7.35] 在视图StudentView1999中将学号为'1600006'的同学记录删除。

```
DELETE  FROM  StudentView1999
WHERE  studentNo='1600006'
```

- 系统将该操作转化为如下的操作：

```
DELETE  FROM  Student
WHERE  year(birthday)=1999     --创建视图中的谓词条件
    AND  studentNo='1600006'
```

7.3.3 视图更新

[例7.36] 在视图SourceView中删除平均成绩大于80分的课程记录。

```
DELETE FROM SourceView
WHERE courseAvg>=80
```

- 对于该操作，数据库管理系统拒绝执行
  - 因为视图SourceView中包含了聚合运算，系统无法将该视图转化为对基本表的操作。
- 一般来讲，如果是行列子集视图，则可以对该视图进行更新操作；其它类型的视图，具体的数据库系统有具体的定义，一般不对其进行更新操作。

7.3.4　删除视图

■ 删除视图：

DROP VIEW <*viewName*> [CASCADE]

- 其中，CASCADE为可选项，选择表示级联删除。
- 该删除视图的语句从数据字典中删除指定的视图
  - 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；
  - 删除基本表时，由该基本表导出的所有视图都必须显式地使用DROP VIEW语句删除。

[例7.37] 删除视图及级联视图。

- 删除视图StudentView1991：

  DROP VIEW StudentView1991

- 级联删除视图SourceView：

  DROP VIEW SourceVIEW CASCADE

视图小结

- 1：掌握视图定义，理解**WITH CHECK OPTION**
- 2：理解视图执行顺序，如果有聚合函数等，如何执行
- 3：掌握视图主要在查询，如果修改，聚合函数等是否可以修改。