第三章 关系数据库标准语言SQL

3.1 SQL概述

- 结构化查询语言（Structured Query Language，SQL），是关系数据库的标准语言，也是一个通用的、功能极强的关系数据库语言
- 目前没有一个数据库系统能够支持 SQL 标准的所有概念和特性
- 许多软件厂商对 SQL 基本命令集还进行了不同程度的扩充和修改，又可以支持标准以外的一些功能特性
- SQL 的特点
    - 综合统一
    - 高度非过程化
    - 面向集合的操作方式
    - 以同一种语法结构提供多种使用方式
    - 语言简洁，易学易用
- 支持 SQL 的关系数据库管理系统同样支持关系数据库三级模式结构，其中外模式包含若干**视图**和部分**基本表**，模式包括若干基本表，内模式包含若干**存储文件**
    - 基本表
        - 本身独立存在的表
        - SQL 中一个关系就对应一个基本表
        - 一个（或多个）基本表对应一个存储文件
        - 一个表可以带若干索引
    - 存储文件
        - 逻辑结构组成了关系数据库的内模式
        - 物理结构对用户是隐蔽的
    - 视图
        - 从一个或几个基本表导出的表
        - 数据库中只存放视图的定义而不存放视图对应的数据
        - 视图是一个虚表
        - 用户可以在视图上再定义视图

3.2 数据定义

- 层次化的数据库对象命名机制
    - 一个关系数据库管理系统的实例中可以建立多个数据库
    - 一个数据库中可以建立多个模式
    - 一个模式下通常包括多个表、视图和索引等数据库对象

- SQL 的数据定义功能

操作对象	创建	删除	修改
模式	`CREATE SCHEMA`	`DROP SCHEMA`	
表	`CREATE TABLE`	`DROP TABLE`	`ALTER TABLE`
视图	`CREATE VIEW`	`DROP VIEW`	
索引	`CREATE INDEX`	`DROP INDEX`	`ALTER INDEX`

3.2.1 模式的定义与删除

1. 定义模式

- 定义模式实际上定义了一个命名空间。在这个空间中可以定义该模式包含的数据库对象，例如基本表、视图、索引等
- SQL 中模式定义语句为

```
CREATE SCHEMA <模式名> AUTHORIZATION <用户名>;
```

- 在`CREATE SCHEMA`中可以接受 `CREATE TABLE`， `CREATE VIEW` 和 `GRANT` 子句，即

```
CREATE SCHEMA <模式名> AUTHORIZATION <用户名> [<表定义子句> | <视图定义子句> | <授权定义子句>]
```

2. 删除模式

- SQL 中删除模式的语句为

```
DROP SCHEMA <模式名> <CASCADE|RESTRICT>;
```

- 其中 `CASCADE` 和 `RESTRICT` 两者必选其一
  - 选择了 `CASCADE` （级联），表示在删除模式的同时把该模式中所有的数据库对象全部删除
  - 选择了 `RESTRICT` （限制），表示如果该模式中定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行。仅当该模式中没有任何下属的对象时才能执行 `DROP SCHEMA` 语句

3.2.2 基本表的定义、删除与修改

1. 定义基本表

SQL 中定义基本表的格式为

```
CREATE TABLE <表名>
       (<列名> <数据类型> [<列级完整性约束条件>]
       [,<列名> <数据类型>[<列级完整性约束条件>]]
       …
       [,<表级完整性约束条件>]);
```

- <表名>：所要定义的基本表的名字
- <列名>：组成该表的各个属性（列）
- <列级完整性约束条件>：涉及相应属性列的完整性约束条件
- <表级完整性约束条件>：涉及一个或多个属性列的完整性约束条件
- 如果完整性约束条件涉及到该表的多个属性列，则必须定义在表级上，否则既可以定义在列级也可以定义在表级

2. 数据类型

- SQL 中域的概念用数据类型来实现
- 定义表的属性时需要指明其数据类型及长度
- 一个属性选用哪种数据类型一般从取值范围和要做哪些运算两方面来考虑

3. 修改基本表

SQL 中修改基本表的格式为

```
ALTER TABLE <表名>
[ADD[COLUMN] <新列名> <数据类型> [完整性约束]]
[ADD <表级完整性约束>]
[DROP [COLUMN] <列名> [CASCADE | RESTRICT]]
[DROP CONSTRAINT <完整性约束名> [RESTRICT | CASCADE]]
[ALTER COLUMN <列名> <数据类型>];
```

- <表名>是要修改的基本表
- `ADD` 子句用于增加新列、新的列级完整性约束条件和新的表级完整性约束条件
- `DROP COLUMN` 子句用于删除表中的列
  - 如果指定了 `CASCADE` 短语，则自动删除引用了该列的其他对象
  - 如果指定了 `RESTRICT` 短语，则如果该列被其他对象引用，关系数据库管理系统将拒绝删除该列
- `DROP CONSTRAINT` 子句用于删除指定的完整性约束条件
- `ALTER COLUMN` 子句用于修改原有的列定义，包括修改列名和数据类型

4. 删除基本表

SQL 删除基本表的格式为

```
DROP TABLE <表名> [RESTRICT| CASCADE];
```

- 若选择 `RESTRICT`，则删除表是有限制的：欲删除的基本表不能被其他表的约束所引用，如果存在依赖该表的对象，则此表不能被删除
- 若选择 `CASCADE`，删除该表没有限制：在删除基本表的同时，相关的依赖对象一起删除

3.2.3 索引的建立和删除

- 建立索引的目的：加快查询速度
  - 由数据库管理员或表的拥有者建立
  - 由关系数据库管理系统自动完成维护
  - 关系数据库管理系统自动使用合适的索引作为存取路径，用户不必也不能显式地选择索引
- 关系数据库管理系统中常见索引：
  - 顺序文件上的索引
  - B+树索引
  - 散列索引
  - 位图索引

1. 建立索引

建立索引的语句格式为

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
ON <表名> (<列名> [<次序>] [,<列名> [<次序>]]…);
```

- <表名>：要建索引的基本表的名字
- 索引：可以建立在该表的一列或多列上，各列名之间用逗号分隔
- <次序>：指定索引值的排列次序，升序为 `ASC`，降序为 `DESC`，缺省值为 `ASC`
- `UNIQUE`：此索引的每一个索引值只对应唯一的数据记录
- `CLUSTER`：表示要建立的索引是聚簇索引

2. 修改索引

对于以建立的索引，如果需要对其重新命名，语句格式为

```
ALTER INDEX <旧索引名> RENAME TO <新索引名>
```

3. 删除索引

删除索引的语句格式为

```
DROP INDEX <索引名>;
```

3.2.4 数据字典

- 数据字典是关系数据库管理系统内部的一组系统表，它记录了数据库中所有定义信息：
  - 关系模式定义
  - 视图定义
  - 索引定义
  - 完整性约束定义
  - 各类用户对数据库的操作权限
  - 统计信息等

- 关系数据库管理系统在执行 SQL 的数据定义语句时，实际上就是在更新数据字典表中的相应信息

3.3 数据查询

SQL 提供了 `SELETE` 语句进行数据查询，该语句具有灵活的使用方式和丰富的功能，其一般格式为

```
SELECT [ALL|DISTINCT] <目标列表达式> [,<目标列表达式>] …
FROM <表名或视图名> [,<表名或视图名>…] | (SELECT语句) [AS] <别名>
[WHERE <条件表达式>]
[GROUP BY <列名1> [HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC | DESC]];
```

- `SELECT` 子句：指定要显示的属性列
- `FROM` 子句：指定查询对象（基本表或视图）
- `WHERE` 子句：指定查询条件
- `GROUP BY` 子句：对查询结果按指定列的值分组，该属性列值相等的元组为一个组。通常会在每组中作用聚集函数。
- `HAVING` 短语：只有满足指定条件的组才予以输出
- `ORDER BY` 子句：对查询结果表按指定列值的升序或降序排序

以下例子均来源于下图的学生-课程数据库

Student

学号 Sno	姓名 Sname	性别 Ssex	年龄 Sage	所在系 Sdept
201215121	李勇	男	20	CS
201215122	刘晨	女	19	CS
201215123	王敏	女	18	MA
201215125	张立	男	19	IS

(a)

Course

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4

(b)

SC

学号 Sno	课程号 Cno	成绩 Grade
201215121	1	92
201215121	2	85
201215121	3	88
201215122	2	90
201215122	3	80

(c)

3.3.1 单表查询

1. 选择表中的若干列

（1）查询指定列

> 例：查询全体学生的学号与姓名

```
SELECT Sno, Sname
FROM Student;
```

（2）查询全部列

> 例：查询全体学生的详细记录

```
SELECT  *
FROM Student;
```

（3）查询经过计算的值

> 例：查询全体学生的姓名、出生年份和所在的院系，要求用小写字母表示系名，且使用列别名改变查询结果的列标题

```
SELECT Sname NAME, 'Year of Birth:' BIRTH, 2014-Sage BIRTHDAY, LOWER(Sdept)
DEPARTMENT
FROM Student;
```

查询结果为：（当前年为2014年）

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1994	cs
刘晨	Year of Birth:	1995	cs
王敏	Year of Birth:	1996	ma
张立	Year of Birth:	1995	is

2. 选择表中的若干元组

（1）消除取值重复的行

> 例：查询选修了课程的学生学号

```
SELECT DISTINCT Sno
FROM SC;
```

（2）查询满足条件的元组

> 例：查询计算机系年龄在20岁以下的学生姓名

```sql
SELECT Sname
FROM  Student
WHERE Sdept = 'CS' AND Sage < 20;
```

3. `ORDER BY` 子句

> 例：查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列

```sql
SELECT  *
FROM  Student
ORDER BY Sdept, Sage DESC;
```

4. 聚集函数

> 例：计算1号课程的学生平均成绩

```sql
SELECT AVG(Grade)
FROM    SC
WHERE Cno= ' 1 ';
```

5. `GROUP BY` 子句

> 例：查询平均成绩大于等于90分的学生学号和平均成绩

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

3.3.2 连接查询

1. 等值与非等值连接查询

> 例：查询每个学生及其选修课程的情况

```sql
SELECT  Student.*, SC.*
FROM    Student, SC
WHERE   Student.Sno = SC.Sno;
```

2. 自身连接

> 例：查询每一门课的间接先修课（即先修课的先修课）

```sql
SELECT  FIRST.Cno, SECOND.Cpno
FROM  Course  FIRST, Course  SECOND #为Course表取两个别名，一个是FIRST，另一个是SECOND
WHERE FIRST.Cpno = SECOND.Cno;
```

3. 外连接

> 例：查询每个学生及其选修课程的情况，保留没有选课的学生

```
SELECT Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade
FROM  Student  LEFT OUT JOIN SC ON (Student.Sno=SC.Sno);
```

4. 多表连接

> 例：查询每个学生的学号、姓名、选修的课程名及成绩

```
SELECT Student.Sno, Sname, Cname, Grade
FROM    Student, SC, Course
WHERE Student.Sno = SC.Sno AND SC.Cno = Course.Cno;
```

3.3.3 嵌套查询

- 一个 `SELECT-FROM-WHERE` 语句称为一个查询块
- 将一个查询块嵌套在另一个查询块的 `WHERE` 子句或 `HAVING` 短语的条件中的查询称为嵌套查询
- 子查询不能使用 `ORDER BY` 子句
- 不相关子查询：子查询的查询条件不依赖于父查询
  - 由里向外，逐层处理。即每个子查询在上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。
- 相关子查询：子查询的查询条件依赖于父查询
  - 首先取外层查询中表的第一个元组，根据它与内层查询相关的属性值处理内层查询，若WHERE子句返回值为真，则取此元组放入结果表
  - 然后再取外层表的下一个元组
  - 重复这一过程，直至外层表全部检查完为止

1. 带有 `IN` 谓词的子查询

> 例：查询选修了课程名为“信息系统”的学生学号和姓名

```
SELECT  Sno,Sname
FROM    Student
WHERE Sno  IN
      (SELECT Sno
       FROM    SC
       WHERE  Cno IN
            (SELECT Cno
             FROM Course
             WHERE Cname= '信息系统'
            )
      );
```

2. 带有比较运算符的子查询

> 例：找出每个学生超过他选修课程平均成绩的课程号

```
SELECT  Sno, Cno
FROM    SC  x
WHERE Grade >= (SELECT AVG (Grade)
                FROM  SC y
                WHERE y.Sno=x.Sno);
```

3. 带有 `ANY`（`SOME`）或 `ALL` 谓词的子查询

> 例：查询非计算机科学系中比计算机科学系任意一个学生年龄小的学生姓名和年龄

```
SELECT  Sname,Sage
FROM    Student
WHERE   Sage <ANY (SELECT  Sage
                   FROM    Student
                   WHERE   Sdept= ' CS ')
AND Sdept <> 'CS ' ;/*注意这是父查询块中的条件 */
```

4. 带有 `EXISTS` 谓词的子查询

> 例：查询所有选修了1号课程的学生姓名

```
SELECT Sname
FROM   Student
WHERE  EXISTS
       (SELECT *
        FROM SC
        WHERE Sno=Student.Sno AND Cno= '1');
```

3.3.4 集合查询

- 集合操作主要包括并操作 `UNION`、交操作 `INTERSECT` 和差操作 `EXCEPT`
- 参加集合操作的各查询结果的列数必须相同；对应项的数据类型也必须相同

> 例：查询选修了课程1或者选修了课程2的学生

```
SELECT Sno
FROM SC
WHERE Cno = '1'
UNION
SELECT Sno
FROM SC
WHERE Cno = '2';
```

> 例：查询计算机科学系的学生与年龄不大于19岁的学生的交集（MySQL目前不支持交操作）

```
SELECT *
FROM Student
WHERE Sdept = 'CS'
INTERSECT
SELECT *
FROM Student
WHERE Sage <= 19
```

实际上就是查询计算机科学系中年龄不大于19岁的学生

```
SELECT *
FROM Student
WHERE Sdept = 'CS' AND Sage <= 19;
```

> 例：查询计算机科学系的学生与年龄不大于19岁的学生的差集（MySQL目前不支持差集操作）

```
SELECT *
FROM Student
WHERE Sdept='CS'
EXCEPT
SELECT  *
FROM Student
WHERE Sage <=19;
```

实际上是查询计算机科学系中年龄大于19岁的学生

```
SELECT *
FROM Student
WHERE Sdept = 'CS' AND Sage > 19;
```

3.3.5 基于派生表的查询

- 子查询不仅可以出现在 `WHERE` 子句中，还可以出现在 `FROM` 子句中，这时子查询生成的临时派生表成为主查询的查询对象
- 如果子查询中没有聚集函数，派生表可以不指定属性列，子查询 `SELECT` 子句后面的列名为其默认属性
- 通过 `FROM` 子句生成派生表时， `AS` 关键字可以省略，但必须为派生关系指定一个别名

> 例：找出每个学生超过他自己选修课程平均成绩的课程号

```
SELECT Sno, Cno
FROM SC, (SELECT Sno, Avg(Grade)
          FROM SC
          GROUP BY Sno)
          AS Avg_sc(avg_sno,avg_grade)
WHERE SC.Sno = Avg_sc.avg_sno and SC.Grade >= Avg_sc.avg_grade
```

3.3.6 `SELECT` 语句的一般格式

`SELECT` 语句的一般格式为

```
SELECT [All|DISTINCT] <目标表达式> [别名] [,<目标表达式>[别名]]…
FROM <表名或视图名> [别名] [,<表名或视图名> [别名]] … | (<SELECT>) [AS] <别名>
[WHERE <条件表达式>]
[GROUP BY <列名1> [HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC|DESC]]
```

- 目标列表达式的可选格式
  - `*`
  - `<表名>. *`
  - `COUNT ([DISTINCT|ALL])`
  - `<表名或视图名> [别名] [,<表名或视图名> [别名]] …`
- 聚集函数的一般格式

$$\left\{\begin{array}{l}\text{COUNT}\\ \text{SUM}\\ \text{AVG}\\ \text{MAX}\\ \text{MIN}\end{array}\right\}([\text{DISTINCT}\,|\,\text{ALL}]\ <\text{列名}>)$$

- `WHERE` 子句的条件表达式的可选格式

1. $<\text{属性列名}>\theta\left\{\begin{array}{l}<\text{属性列名}>\\ <\text{常量}>\\ [\text{ANY}\,|\,\text{ALL}]\ (\text{SELECT 语句})\end{array}\right\}$
2. $<\text{属性列名}>[\text{NOT}]\ \text{BETWEEN}\left\{\begin{array}{l}<\text{属性列名}>\\ <\text{常量}>\\ (\text{SELECT 语句})\end{array}\right\}\ \text{AND}\ \left\{\begin{array}{l}<\text{属性列名}>\\ <\text{常量}>\\ (\text{SELECT 语句})\end{array}\right\}$
3. $<\text{属性列名}>[\text{NOT}]\ \text{IN}\left\{\begin{array}{l}(<\text{值1}>\ [,<\text{值2}>\ldots])\\ (\text{SELECT 语句})\end{array}\right\}$
4. `<属性列名> [NOT] LIKE <匹配串>`
5. `<属性列名> IS [NOT] NULL`
6. `[NOT] EXISTS (SELECT语句)`
7. $<\text{条件表达式}>\left\{\begin{array}{l}\text{AND}\\ \text{OR}\end{array}\right\}<\text{条件表达式}>\left(\left\{\begin{array}{l}\text{AND}\\ \text{OR}\end{array}\right\}<\text{条件表达式}>\ldots\right)$

3.4 数据更新

3.4.1 插入数据

1. 插入元组

插入元组的 `INSERT` 语句的格式为

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

- 对于 `INTO` 子句
  - 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
  - 没有指定属性列时，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
  - 指定部分属性列时，插入的元组在其余属性列上取空值
- 对于 `VALUES` 子句，提供的值的个数和值的类型必须与 `INTO` 子句匹配

> 例：将一个新学生元组插入到 `Student` 表中

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

2. 插入子查询结果

插入子查询结果的 `INSERT` 语句格式为

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

> 例：对每一个系，求学生的平均年龄，并把结果存入数据库

首先在数据库中建立一个新表，其中一列存放系名，另一列存放相应的学生平均年龄

```
CREATE TABLE Dept_age
            (Sdept CHAR(15)
             Avg_age SMALLINT)
```

然后对 `Student` 按系分组求平均年龄，再把系名和平均年龄存入新表中

3.4.2 修改数据

修改操作又称为更新操作，其语句的一般格式为

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

- 其功能是修改指定表中满足 `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

> 例：将计算机科学系全体学生的成绩置零

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

3.4.3 删除数据

删除语句的一般格式为

```
DELETE
FROM <表名>
[WHERE <条件>];
```

- `DELETE` 语句的功能是从指定表中删除满足 `WHERE` 子句条件的所有元组
- 如果省略 `WHERE` 语句则表示删除表中的所有元组，但表的定义仍在字典中
- `DELETE` 语句删除的是表中的数据，而不是关于表的定义

> 例：删除计算机科学系所有学生的选课记录

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

3.5 空值的处理

- 空值就是“不知道”或“不存在”或“无意义”的值
- SQL 语言允许某些元组的某些属性在一定情况下取空值，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

1.空值的产生

> 例：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);
```

2.空值的判断

> 例：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;
```

3.空值的约束条件

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，码属性不能取空值

4.空值的算数运算、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

- 在查询语句中，只有使 `WHERE` 和 `HAVING` 子句的选择条件为 `TRUE` 的元组才被选出作为输出结果

> 例：选出选修1号课程的不及格的学生以及缺考的学生

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

/*或者*/

SELECT Sno
FROM SC
WHERE Grade < 60 AND Cno = '1'
UNION
SELECT Sno
FROM SC
WHERE Grade IS NULL AND Cno = '1'
```

3.6 视图

- 视图是从一个或几个基本表（或视图）导出的虚表
- 数据库中只存放视图的定义，而不存放视图对应的数据
- 一旦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

3.6.1 定义视图

3.6.1.1 建立视图

SQL 语言用 `CREATE VIEW` 命令建立视图，其一般格式为

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

- 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，则取决于具体系统的实现
- `WITH CHECK OPTION` 表示对视图进行 `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）
- 组成视图的属性列名或者全部省略或者全部指定
- 如果省略了视图的各个属性列名，则隐含该视图由子查询中 `SELECT` 子句目标列中的诸字段组成
- 下列情况必须指定组成视图的所有列名
  - 某个目标列是聚集函数或列表达式
  - 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

> 例：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;
```

3.6.1.2 删除视图

删除视图的语句格式为

```
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

3.6.2 查询视图

关系数据库管理系统执行对视图的查询时，首先进行有效性检查，检查查询中涉及的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义，把定义中的子查询和用户的查询结合起来，转换成等价的对基本表的查询，然后在执行修正了的查询。这一转换过程称为**视图消解**。

> 例：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';
```

3.6.3 更新视图

- 由于视图是不实际存储数据的虚表，因此对视图的更新最终要转换成对基本表的更新
- 像查询视图一样，对视图的更新操作也是通过视图消解，转换为对基本表的更新操作
- 一般地，行列子集视图是可更新的
- 对其他类型视图的更新不同系统有不同限制。一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新

> 例：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';
```

转换后的更新语句为：

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

3.6.4 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，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
- 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