数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

第三章 关系数据库标准语言SQL

Principles of Database Systems

第三章 课堂练习

1. 关于SQL语言，下列说法正确的是（）。

A. 数据控制功能不是SQL语言的功能之一。

B. SQL采用的是面向记录的操作方式，以记录为单位进行操作。

C. √ SQL是非过程化语言，用户无须指定存储路径。

D. SQL 作为嵌入式语言，语法与独立式的语言有较大的差异。

2. 设关系R（ABC）包含的数据如右图所示：

```
SELECT * FROM R
    WHERE C IN   (SELECT B FROM R);
```

A	B	C
30	20	NULL
10	NULL	30

该SQL语句的查询结果为：

A. √ NULL

B. (30,20,NULL)

3.4.4 集合查询

- 集合操作的种类
  - 并操作UNION
  - 交操作INTERSECT
  - 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

3.4.4 集合查询

[例] 查询计算机科学系的学生及年龄不大于19岁的学生。

方法一：

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

方法二：

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

- UNION：将多个查询结果合并起来时，系统自动去掉重复元组。
- UNION ALL：将多个查询结果合并起来时，保留重复元组

3.4.4 集合查询

[例] 查询选修课程1的学生集合与选修课程2的学生集合的交集

方法一：

```
SELECT Sno
  FROM SC
  WHERE Cno=' 1 '
  INTERSECT
  SELECT Sno
  FROM SC
  WHERE Cno='2 '
```

方法二：

```
SELECT Sno FROM SC
    WHERE Cno=' 1 ' AND Sno IN
            (SELECT Sno
              FROM SC
              WHERE Cno=' 2 ');
```

3.4.4 集合查询

[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。

方法一：

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

方法二：

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

3.4.5 基于派生表的查询

- 当子查询出现在FROM子句中，这时子查询生成临时派生表（derived table）成为主查询的查询对象。

[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, (SELECT Sno, Avg(Grade) FROM SC
                Group by Sno)
              AS Avg_sc(avg_sno, avg_grade)
Where SC.sno = avg_sno
      AND grade > avg_grade;
```

必须为派生关系指定别名

总结：SELECT语句的分解

- 单表查询：

```
SELECT ……
FROM  t1
WHERE ……
……
```

- 多表连接查询：

```
SELECT ……
FROM  t1，t2, …
WHERE ……
 t1.col1 谓词 t2.col2
```

- 集合查询：

```
SELECT 查询块
集合操作
SELECT 查询块
```

- 派生表查询：

```
SELECT …
FROM (SELECT 子
查询) ……
```

- 嵌套查询：

```
SELECT ……
FROM ……
WHERE
   [ col1 谓词（SELECT查询块）]
   [ | EXISTS（ SELECT查询块）]
   ……
```

```
SELECT ……
FROM ……
WHERE …
GROUP BY …
HAVING
 [ col1 谓词 （SELECT查询块）]
 [ | EXISTS（SELECT查询块）] ……
```

SELECT综合实例

假设银行数据库关系模式为：

银行： Branch=(Bname, city, assets)

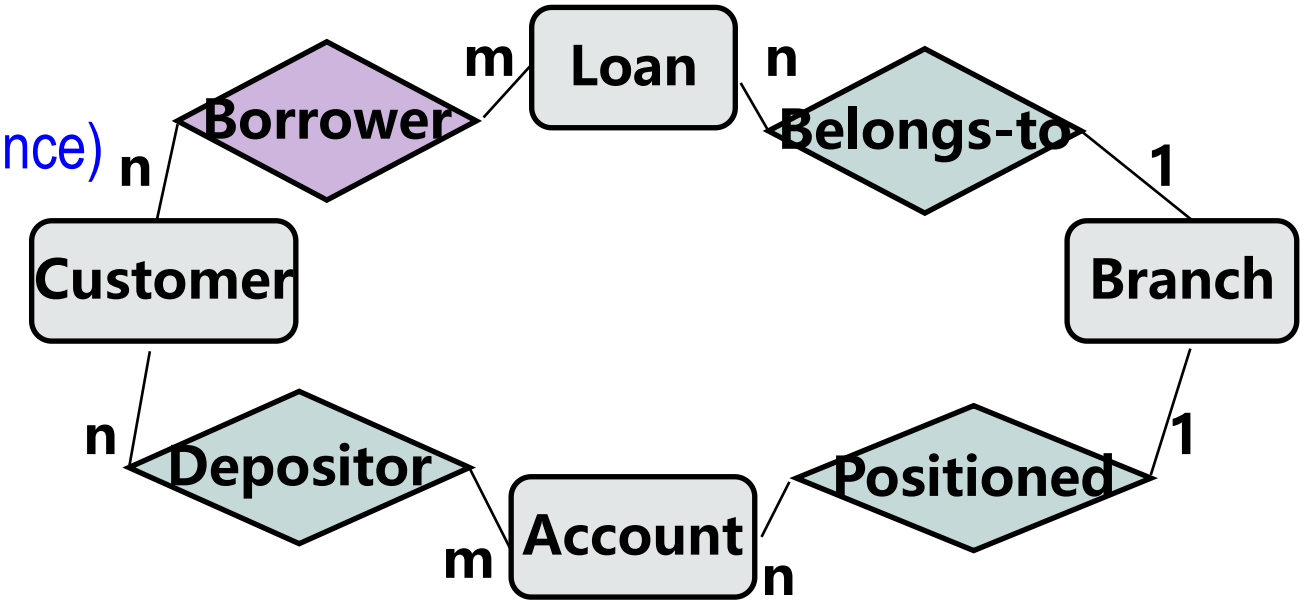
顾客： Customer=(Cno, Cname, street, city)

贷款： Loan=(Lno, Bname, amount)

借贷： Borrower=(Cno, Lno)

账户： Account=(Ano, Bname, balance)

存款： Depositor=(Cno, Ano)



SELECT综合实例

银行数据库关系模式为：

银行 Branch=(Bname, Bcity, Bassets)
顾客 Customer=(Cno, Cname, Cstreet, Ccity)
贷款 Loan=(Lno, Bname, amount)
借贷 Borrower=(Cno, Lno)
账户 Account=(Ano, Bname, balance)
存款 Depositor=(Cno, Ano)

- 思考题：

1）找出在“Perry”银行有贷款的客户姓名及贷款数；
2）找出资产至少比位于Brooklyn的某一家支行高的支行名；
3）找出银行中在Perry银行既有贷款又有账户的客户姓名；
4）找出平均余额最高的支行；
5）找出住在Harrison且在银行中至少有三个账户的客户的平均余额；
6）找出在Brooklyn的所有支行都有账户的客户；

SELECT综合实例

银行数据库关系模式为：

Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

□ 思考题解答：

1、找出在“Perry”银行有贷款的客户姓名及贷款数；

解法一：整体法 关键词（客户、有贷款、贷款）

1）找from　customer ⋈ borrower ⋈ loan

2）用where过滤　Bname="perry"

3）用select投影　Cname,amount

```
SELECT  Cname,amount
    FROM   customer C, borrower B, loan L
    WHERE  C.Cno=B.Cno and B.Lno = L.Lno and Bname="perry"
```

SELECT综合实例

1、找出在“Perry”银行有贷款的客户姓名及贷款数；

□ 解法二：分步法

1）找出在perry有贷款的所有Cno集合

用一个select块从borrower ⋈ loan 中查询；

2）从customer中选择其Cno IN 1）结果集的元组

```
select  Cname,amount
    from   customer C
    where  C.Cno  IN   (select  Cno
                        from    borrower B, loan L
                        where  B.Lno = L.Lno
                           and Bname="perry");
```

银行数据库关系模式为：

Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

- 解法三：相关法

1）用exists函数构造一个能够判断任意一个顾客元组t，t.Cno是否在perry银行有贷款集合中的逻辑函数；

2）对customer中每个元组，调用exists判断

```
select  Cname,amount
      from   customer C
      where  EXISTS  (select  *
                      from   borrower B, loan L
                      where  C.Cno = B.Cno
                             and B.Lno = L.Lno
                             and Bname="perry");
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

- 思考题解答：

2、找出资产至少比位于Brooklyn的某一家支行多的支行名；

解法一：整体法 关键词（一家支行、位于Brooklyn的支行）

1）找from branch(角色T) × branch(角色S)

2）用where过滤 T.assets>S.assets and S.city="Brooklyn"

3）用select投影 T.Bname

```
select  T.Bname
   from   branch T, branch S
   where  T.assets > S.assets
              and S.city = 'Brooklyn';
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

2、找出资产至少比位于Brooklyn的某一家支行多的支行名；

解法二：分步法

1）找出位于Brooklyn的支行的总资产集合，用一个select块从branch中查询；

2）对上述结果用集合函数any()求出集合的任意一个

3）从branch中选择其总资产大于any()函数返回值的元组

```
select  Bname
from   branch
where  assets > any(select  assets
                    from   branch
                    where  city='Brooklyn');
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

- 思考题解答

3、找出银行中在Perry银行既有贷款又有账户的客户姓名；

解法一：使用交运算

1）找出在Perry银行有贷款的客户

2）找出在Perry银行有账户的客户

3）用intersect求交集

```
(select  distinct Cno
 from   borrower B, loan L
 where  B.Lno=L.Lno and Bname="Perry")
  intersect
(select  distinct Cno
 from   depositor D, account A
 where  D.Ano=A.Ano and Bname="Perry");
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

3、找出银行中在Perry银行既有贷款又有账户的客户姓名；

□ 解法二： 1）找出在Perry银行有贷款的客户（集合S1）

2）找出在Perry银行有账户的客户（集合S2）

3）对于集合S1中每个元组，判断是否属于S2

```
select  distinct Cno
from    borrower, loan
where  B.Lno=L.Lno and Bname="Perry"
          and Cno IN (select  distinct Cno
                              from   depositor, account
                              where  D.Ano=A.Ano
                                        and Bname="Perry");
```

或 (Bname,Cno) IN (select distinct Bname,Cno
from depositor, account
where D.Ano=A.Ano);

SELECT综合实例

3、找出银行中在Perry银行既有贷款又有账户的客户姓名；

- 解法三：相关法

1）用exists函数构造一个能够判断任意一个顾客元组t，t.Cno是否在perry银行有账户的逻辑函数；

2）对borrower,loan中每个在perry银行有贷款元组，调用exists判断该元组是否有账户

```
select  Cname
    from   borrower B, loan L
    where  L.Bname=’  Perri' and B.Lno=L.Lno and
           EXISTS  (select  *
                    from   depositor D, account A
                    where  B.Cno = D.Cno
                           and D.Ano = A.Ano
                           and A.Bname="perry");
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

- 思考题解答

4、找出平均余额最高的支行；

分步法：

1）找出每个银行的平均余额集合，用一个select块和集函数avg()从account中查询；

2）对上述结果用集合函数all()求出平均余额的所有

3）从account中选择平均余额大于all()函数返回值的元组

```
select  Bname
from   account
group by  Bname
having (avg(balance) >=all(select  avg(balance)
                           from   account
                           group by  Bname));
```

SELECT综合实例

- 思考题解答

5、找出住在Harrison且在银行中至少有三个账户的客户的平均余额；

解法：整体法 关键词（客户、有账户、账户余额）

1）找from customer ⋈ depositor ⋈ account

2）用where过滤 Ccity=“Harrison” ，得出住在Harrison且在银行中有账户的客户

3）用group分组 求出每个Harrison客户的账户数

4）用having 对每个分组过滤

```
select  C.Cno, avg(balance)
from   customer C, depositor D, account A
where  C.Cno=D.Cno and D.Ano = A.Ano  and C.city="Harrison"
group by D.Cno
having  count(D.Ano)>=3)
```

银行数据库关系模式为：

Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

SELECT综合实例

银行数据库关系模式为：

Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

6、找出在Brooklyn的所有支行都有账户的客户;

方法一：双重否定。

```
select  distinct S.Cno
from   Depositer S
where   not exists ( select  *
                from   Branch B
                where   Bcity=  'Brooklyn'
               and not exists
                    ( select  *
                      from   depositor T, account R
                      where   T.Ano=R.Ano
                              and  S.Cno=T.Cno
                              and B.Bname= R.Bname ));
```

不存在Brooklyn的某家支行，该客户在该支行没有账户

SELECT综合实例

6、找出在Brooklyn的所有支行都有账户的客户;

方法二：集合A包含集合B 等价于 not exists (B - A);

```
select  distinct  Cno
   from   depositor S
   where   not exists (( select  Bname
                         from   branch
                         where   Bcity='Brooklyn')
                         except
                        ( select  Bname
                        from   depositor T, account R
                        where   T.Ano=R.Ano
                                and  S.Cno=T.Cno));
```

不存在Brooklyn的某家支行，该客户在该支行没有账户

SQL随堂练习

考虑下面关系数据库： 公司 com (cno, cname， city)

员工 emp (eno, ename, street, city, mno)

;mno 是外键，引用emp(eno)，表示该员工的上级经理

工作 works (eno, cno, salary)

请用关系代数表达式表示下面查询?

1）找出公司（编号为1001）的的所有员工姓名。

2）找出公司名为“DM”的所有收入在10000元以上员工的姓名和居住城市。

3）找出所有居住地与工作的公司在同一个城市的员工姓名。

4）找出与其经理居住在同一个城市的所有员工姓名和他的经理姓名。

5）找出“DM”公司中不在其分公司（编号为1001）工作的所有员工编号和姓名。

6）找出比公司（编号为1002）的所有员工收入都高的公司名和员工姓名。

7）假设公司可以在几个城市部署分部(cname相同，但cno不同)。找出在“DM”公司所在的各个城市都有分部的公司。

第三章 关系数据库标准语言SQL

3.5 数据更新

3.5.1 插入数据

插入数据是把新的记录插入到一个存在的表中。

1. 元组值的插入

语法：

```
INSERT INTO 基本表[(列名[, 列名]...)] VALUES(元组值);
```

作用：将一条元组值插入到表中。

注意：

- 列名的排列顺序不一定和表定义时的顺序一致。
- INTO子句中指定列名时，元组值的字段排列顺序必须和列名的排列顺序一致、个数相等、数据类型匹配。
- INTO子句中未指定列名时，元组值必须在每个属性列上均有值，且值的排列顺序与表中属性列的排列顺序一致。

3.5.1 插入数据

【例1】往基本表SC中插入一个元组值

INSERT INTO SC VALUES('S004','C011',90);

【例2】往基本表SC中插入部分元组值

INSERT INTO SC(sno, cno) VALUES('S004','C025');

注：对于INTO子句中没有出现的列，则新插入的记录在这些列上将取空值或缺省值，如上例的grade字段即赋空值。但有NOT NULL约束的属性列不能取空值。

Mysql

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES
(value1, value2, value3, ...),
(value1, value2, value3, ...),
...; //插入多列
```

3.5.1 插入数据

2. 查询结果的插入

将一个表中的数据抽取若干行插入另一表中，可以通过子查询来实现。

语法：

```
INSERT INTO <基本表名> [(<列名表)] 子查询;
```

作用：将子查询返回的结果数据集插入到表中。其功能是批量插入。

要求：查询语句的目标列必须与INTO子句匹配。

3.5.1 插入数据

【例3】将信息系（IS）平均成绩大于80分的学生学号和平均成绩存入另一个基本表S_GRADE(Sno, Avg_grade)。

```
INSERT INTO  S_GRADE (Sno, Avg_grade)
SELECT Sno, AVG(grade)
        FROM  SC
        WHERE Sno IN
              ( SELECT Sno
                FROM STUDENT
                WHERE sdept = 'IS')
        GROUP BY sno
        HAVING AVG(grade)>80;
```

3.5.1 插入数据

RDBMS在执行插入／更新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

3.5.2 删除数据

语法：DELETE FROM <表名> [WHERE 条件表达式]

作用：从表中删除符合WHERE子句中删除条件的元组；若WHERE子句缺省，则表示要删除表中的所有元组。

【例4】删除学号为 ‘S001’的学生信息。

```
DELETE FROM STUDENT WHERE sno='S001';
```

【例5】删除所有学生信息。

```
DELETE FROM STUDENT;
```

不带条件的DELETE语句与DROP TABLE语句的区别：

- DROP TABLE语句删除表和表中的所有数据；
- DELETE语句只删除表中的数据，表仍然保留。

3.5.2 删除数据

□ 带子查询的删除语句

子查询同样也可以嵌套在DELETE语句中，用以构造执行删除操作的条件。

【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');
```

3.5.3 更新数据

- 语法：

```
UPDATE  <基本表名>
  SET <列名> = 值表达式 [, <列名> = 值表达式…]
  [WHERE  条件表达式]
```

- 作用：对表中满足WHERE条件的元组，按SET子句修改相应列的值。若WHERE子句缺省，则表示对所有元组进行修改。

【例5】把所有学生的年龄加1。

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

【例6】把课程号为 ‘C5’的课程名改为“电子商务”。

```
UPDATE  COURSE  SET Cname='电子商务'  WHERE Cno = 'C5';
```

3.5.3 更新数据

□ 带子查询的修改语句

子查询也可以嵌套在UPDATE语句中，用以构造执行修改操作的条件。

例 将计算机科学系全体学生的成绩加10分。

```
UPDATE SC
SET grade=grade+10
WHERE  Sno  IN  ( SELETE Sno
                  FROM Student
                  WHERE Sdept='CS');
```

修改操作与数据库的一致性

UPDATE语句一次只能操作一个表。这会带来一些问题。

例如，学号为95007的学生因病休学一年，复学后需要将其学号改为96089，由于Student表和SC表都有关于95007的信息，因此两个表都需要修改，这种修改只能通过两条UPDATE语句进行。

- 第一条UPDATE语句修改Student表：

```
UPDATE Student
    SET Sno='96089'
    WHERE Sno='95007';
```

- 第二条UPDATE语句修改SC表：

```
UPDATE SC
    SET Sno='96089'
    WHERE Sno='95007';
```

必须保证这两条UPDATE语句要么都做，要么都不做。

事务

3.7 视图

视图是从一个或几个基本表（或视图）导出的一个虚表。

- 数据库中只存放视图的定义而不存放视图的数据，这些数据仍放在原来的基表中。当基表中的数据发生变化时从视图中查出的数据也随之改变了。
- 视图一经定义就可以对其进行查询，但对视图的更新操作有一定的限制。

3.7.1 视图的定义

1. 建立视图

语法：

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

3.7.1 视图的定义

说明：

视图的列名为可选项，缺省时，其列名即为子查询的查询目标列。但是在以下情况下，视图列名不可省略：

- 视图由多个表连接得到，在不同的表中存在同名列，则需指定列名；
- 当视图的列名为表达式或集函数的计算结果时，而不是单纯的属性名时，则需指明列名。
- 在子查询中不许使用ORDER BY子句，如果需要排序，则可在视图定义后，对视图查询时再进行排序。
- 如果指明了WITH CHECK OPTION选项，则在对视图进行更新时，所影响的元组必须满足视图定义中的WHERE条件。

3.7.1 视图的定义

- 行列子集视图：从一个基本表中导出，只是去掉了某些行或列(保留原表的主码)，这样的视图称为行列子集视图。
- 多表视图：从多个基本表或视图中导出。
- 带表达式的视图，即带虚拟列的视图。
- 分组视图，子查询带集函数和GROUP BY分组的视图。

3.7.1 视图的定义

【例1】建立计算机学院（CS）学生视图，并要求进行修改和插入操作时仍需保证该视图只有CS系的学生（单表视图）。

```
CREATE VIEW CS_STUDENT AS
        SELECT sno, sname
        FROM  STUDENT
        WHERE sdept = 'CS'
        WITH CHECK OPTION;
```

【例2】建立计算机学院选修5号课程的学生视图（多表视图）。

```
CREATE VIEW CS_S1(sno,sname,grade) AS
        SELECT STUDENT.sno,sname,grade
        FROM  STUDENT, SC
        WHERE sdept='CS'  AND
        STUDENT.sno=SC.sno  AND  cno='5';
```

3.7.1 视图的定义

【例3】建立计算机学院选修了5号课程且成绩在90分以上的学生视图（视图之上建视图）。

```
CREATE VIEW CS_S2 AS
        SELECT sno, sname, grade
        FROM  CS_S1
        WHERE grade >= 90;
```

【例4】建立学生出生年份的视图（表达式视图）。

```
CREATE VIEW student-year (sno, sname, sbirth)
AS
Select sno，  sname，  1999-sage
From student;
```

3.7.1 视图的定义

【例5】求学生平均成绩视图（分组视图）。

```
CREATE VIEW Sc-AVG(SNO,FAVG)
AS
Select sno， AVG(grade)
from SC
Group By Sno;
```

【例6】将Student表中所有女生记录定义为一个视图（不指定属性列）

```
CREATE VIEW F_Stu(F_Sno,name,sex,age,dept)
 AS
 SELECT *  FROM  Student
 WHERE Ssex='女';
```

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

3.7.1 视图的定义

2. 删除视图

□语法：

```
DROP  VIEW <视图名>  [CASCADE] ;
```

□作用：从数据库中删除一个视图的定义信息。

CASCADE表示把该视图和它导出的视图一起删除。

【例】撤消视图CS_S2。

```
DROP VIEW CS_S2;
```

注：视图删除后，只会删除该视图在数据字典中的定义，而与该视图有关的基本表中的数据不会受任何影响。

3.7.2 查询视图

- 用户角度：查询视图与查询基本表相同
- DBMS执行对视图的查询时，
  - 首先进行有效性检查，检查查询涉及的表、视图等是否在数据库中存在；
  - 如果存在，则从数据字典中取出查询涉及的视图的定义；
  - 把定义中的子查询和用户对视图的查询结合起来，转换成等价的基本表的查询；
  - 然后再执行这个经过修正的查询。

将对视图的查询转换为对基本表的查询的过程称为视图的消解（View Resolution）。

3.7.2 查询视图

例：查询计算机学院学习了5号课程且成绩为95分的学生学号和姓名。

方案1：从基本表中查询。

```
SELECT STUDENT.sno,sname
FROM  STUDENT, SC
WHERE STUDENT.sno=SC.sno AND sdept='CS'
AND cno='5' AND grade=95 ;
```

方案2：利用视图CS_S1查询。

```
SELECT  sno, sname
FROM CS_S1
WHERE  grade=95;
```

```
CREATE VIEW CS_S1(sno,sname,grade) AS
    SELECT STUDENT.sno,sname,grade
    FROM  STUDENT, SC
    WHERE sdept='CS'  AND
    STUDENT.sno=SC.sno  AND  cno='5';
```

由此可见，利用视图可以简化复杂的查询语句。

3.7.2 查询视图

- 视图消解法的局限
  - 有些情况下，视图消解法不能生成正确查询，需要特殊判断处理。

[例]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;
```

S_G视图的子查询定义：

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

查询执行转换：

错误：

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

正确：

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

3.7.3 更新视图

- 视图的更新操作包括插入、修改和删除数据，其语法格式如同对基本表的更新操作一样。
- 由于视图是一张虚表，所以对视图的更新，最终实际上是转换成对基本表的更新。

[例] 对视图CS_STUDENT的更新：

```
INSTER  INTO CS_STUDENT VALUES (‘04008’ , ‘张立’  );
```

在执行时将转换为对表STUDENT的更新：

```
INSERT INTO STUDENT
                VALUES (‘04008’, ‘张立’  , NULL, NULL, NULL);
```

3.7.3 更新视图

[例] 向信息系学生视图IS_S中插入一个新的学生记录：200215129，赵新，20岁。

```
INSERT
INTO IS_Student
VALUES(‘95029’，  ‘赵新’  ，20);
```

转换为对基本表的更新：

```
INSERT
INTO  Student(Sno，Sname，Sage，Sdept)
VALUES(‘200215129 ’，'赵新'，20，'IS' );
```

3.7.3 更新视图

- 更新视图的限制：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

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

例：视图S_G为不可更新视图。

```
UPDATE  S_G
SET    Gavg=90
WHERE  Sno= '200215121';
```

这个对视图的更新无法转换成对基本表SC的更新。

3.7.3 更新视图

视图更新的约束：

根据视图的定义可将视图分为可更新视图和不允许更新的视图。如下列情况的视图为不允许更新的视图：

- 字段由表达式或常数组成，不允许执行INSERT和该字段上的UPDATE，但可以执行DELETE；
- 字段由集函数组成，不允许更新；
- 视图定义含有GROUP BY或DISTINCT，不允许更新；
- 有嵌套查询，且内外查询使用相同表时，不允许更新；
- 定义中有多表联接时，不允许更新；
- 由不允许更新的视图导出的视图，不允许更新。

3.7.3 更新视图

视图的作用：

- 简化用户的操作
- 使用户能以多种角度看待同一数据
- 对重构数据库提供一定程度的逻辑独立性
- 是数据共享与保密的一种安全机制
- 适当的利用视图可以更清晰的表达查询

课堂练习

1.（多选题）下列说法不正确的是（ ）

A. 基本表和视图一样，都是关系

B. √ 可使用SQL对视图进行操作，视图都可以进行插入

C. 可以从多个基本表或视图上定义视图

D. √ 基本表和视图中都存储数据

小结

- SQL语言概述（SQL与关系代数、关系演算是等价的；SQL的5大特点）。
- SQL语言具有数据定义、数据查询、数据更新、数据控制4大功能。数据定义语句（请注意与关系数据库的理论定义（数据结构、完整性约束）相对照。数据查询功能最为丰富和复杂。
- 进一步介绍了索引和视图的概念及其作用。
- SQL学习，应反复上机加强练习。
  - MySQL 8.0
- 本章作业：P130 4，5，9