

第三章 关系数据库标准语言SQL (续)



3.4 数据查询

3.4.1 单表查询

3.4.2 连接查询

3.4.3 嵌套查询

3.4.4 集合查询

3.4.5 **Select**语句的一般形式



嵌套查询的应用场景

- 一个查询块是对关系集合进行搜索找出满足资格的元组。对目标关系中成员 t 的判断可能会出现情况：

(1) 该成员 t 是否属于某个select结果集合 R ;

$t \in R$ 是否成立? (属于运算)

(2) 该成员是否比某个select集中所有成员或至少一个成员大或小;

$t > R$ 中所有或某个成员是否成立? (比较运算)

(3) 该成员是否能使得集合逻辑式成立;

t 是否能使得逻辑命题 L 成立? 其中: L 是一个通过 t 求出的集合逻辑式。 $L(R(t))$ (逻辑运算)



➤图为学生-课程数据库中的student关系、Course关系、SC关系

Course:

Cno	Cname	Cpno	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

SC:

Sno	Cno	Grade
200215121	1	92
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80

Student:

Sno	Sname	Ssex	Sage	Sdept
200215121	李勇	男	20	CS
200215122	刘晨	女	19	IS
200215123	王敏	女	18	MA
200215125	张立	男	19	IS



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'
        )
    );
```

③ 最后在Student关系中
取出Sno和Sname

② 然后在SC关系中找到选
修了3号课程的学生学号

① 首先在Course关系中
找出“信息系统”的课
程号，为3号



嵌套查询求解方法

- 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。



嵌套查询求解方法

- 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



情形二、带有比较运算符的嵌套子查询

[例] 找出每个超过他选修课程平均成绩的学生学号及课程号。

分析：

- 1) 用一个select块构造一个能够查询某个元组t的平均成绩的子函数
- 2) 从SC中搜索出满足其成绩大于该元组平均成绩的选课记录

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

相关子查询



带有比较运算符的子查询（续）

- 可能的执行过程：

1. 从外层查询中取出SC的一个元组x，将元组x的Sno值（200215121）传送给内层查询。


```
SELECT AVG(Grade)
FROM SC y
WHERE y.Sno='200215121';
```

2. 执行内层查询，得到值88（近似值），用该值代替内层查询，得到外层查询：

```
SELECT Sno, Cno
FROM SC x
WHERE Grade >=88;
```

3. 执行这个查询，得到 (200215121, 1) (200215121, 3)
4. 外层查询取出下一个元组重复做上述1至3步骤，直到外层的SC元组全部处理完毕。结果为：

(200215121, 1) (200215121, 3) (200215122, 2)



➤图为学生-课程数据库中的student关系、Course关系、SC关系

Course:

Cno	Cname	Cpno	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

SC:

Sno	Cno	Grade
200215121	1	92
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80

Student:

Sno	Sname	Ssex	Sage	Sdept
200215121	李勇	男	20	CS
200215122	刘晨	女	19	IS
200215123	王敏	女	18	MA
200215125	张立	男	19	IS

带有比较运算符的子查询问题分析

问题：在嵌套查询情形二中，若需要判断关系中元组t与一个集合的“任意一个”或“所有”是否满足某个关系式，该如何解决？

解决：SQL中给出了两个特殊聚集函数ANY(SOME)、ALL，能够求出一个集合的“任意一个”或“所有”元组

谓词语法：

- ANY (R) : R的任意一个值
- ALL (R) : R所有值

谓词语义：

与关系运算符配合使用

> any(R), 表示“至少比R...的某一个大” ;

> all(R), 表示“比R...所有大” ;




带有ANY或ALL谓词的子查询示例

[例] 查询其他系中比计算机科学**某一学生年龄**小的学生姓名和年龄

分析：1) 用一个select找出计算机专业所有学生年龄集合R;
2) 对R做any运算, any(R)为R中任意一个;
3) 从学生表中找出满足“非计算机专业”且其年龄小于any(R)的元组

```
SELECT Sname, Sage
FROM Student
WHERE Sage < ANY (SELECT Sage
                   FROM Student
                   WHERE Sdept= 'CS')
AND Sdept <> 'CS' ;
```



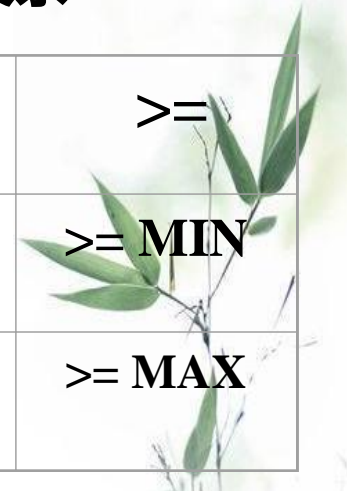
带有ANY或ALL谓词的子查询示例

- 上例可以用聚集函数实现

```
SELECT Sname, Sage
FROM Student
WHERE Sage < (SELECT MAX(Sage)
              FROM Student
              WHERE Sdept= 'CS ')
AND Sdept <> 'CS' ;
```

表 ANY、ALL谓词与聚集函数、IN谓词的等价转换关系

	=	<>或!=	<	<=	>	>=
ANY	IN	--	<MAX	<=MAX	>MIN	>= MIN
ALL	--	NOT IN	<MIN	<= MIN	>MAX	>= MAX



带有ANY或ALL谓词的子查询示例

[例] 查询其他系中比计算机科学系所有学生年龄都小的学生姓名及年龄。

方法一：用ALL谓词

```
SELECT Sname, Sage
FROM Student
WHERE Sage < ALL
      (SELECT Sage
       FROM Student
       WHERE Sdept= ' CS ')
AND Sdept <> ' CS ';
```



带有ANY或ALL谓词的子查询示例

方法二：用聚集函数

```
SELECT Sname, Sage
FROM Student
WHERE Sage <
      (SELECT MIN(Sage)
       FROM Student
       WHERE Sdept= ' CS ')
AND Sdept <>' CS ';
```



带有**ANY**或**ALL**谓词的子查询（续）

- 用集函数实现子查询通常比直接用**ANY**或**ALL**查询效率要高，因为前者通常能够减少比较次数
- 原因：集函数首先把需要比较的集合计算出来（通常情况下是变小的），缩小了和父查询比较的次数



情形三、带有逻辑表达式的嵌套子查询

- **EXISTS**: 本质上是一个返回值为“真”或“假”的集函数, 用于判断一个集合是否为空。
 - 若内层查询结果非空, 则外层的WHERE子句返回真值
 - 若内层查询结果为空, 则外层的WHERE子句返回假值
- **NOT EXISTS**: 用于判断一个集合是否不为空。
 - 若内层查询结果非空, 则外层的WHERE子句返回假值
 - 若内层查询结果为空, 则外层的WHERE子句返回真值



带有EXISTS谓词的子查询示例

[例] 查询所有选修了2号课程的学生姓名。

分析：

- 1) 对SC使用一个select块编写一个对参数Student.Sno，求其选2号课的选课记录集合R；
- 2) 再对R使用集合逻辑函数EXISTS，用于判断Student.Sno是否选过2号课
- 3) 对Student中每个学生元组t，选择出能够让逻辑函数EXISTS(R)成立的元组

```
SELECT Sname  
FROM Student  
WHERE EXISTS  
    (SELECT *  
        FROM SC  
        WHERE Sno=Student.Sno AND Cno= ' 2 ');
```



带有EXISTS谓词的子查询

■ 带EXISTS谓词查询的执行过程

外	Sno	Sname	Ssex	Sage	Sdept	内	Sno	Cno	Grade
→	0215121	李勇	男	20	CS	→	0215121	1	92
	0215122	刘晨	女	19	IS	→	0215121	2	85
	0215123	王敏	女	18	MA		0215126	2	88
	0215125	张立	男	18	IS		0215122	2	90
							0215122	3	80

➤ 用带EXISTS谓词的嵌套查询

SELECT Sname FROM Student
WHERE EXISTS

(SELECT * FROM SC

WHERE Sno=Student.Sno AND Cno= '2')

Sname
李勇

带有EXISTS谓词的子查询

■ 带EXISTS谓词查询的执行过程：

外	Sno	Sname	Ssex	Sage	Sdept	内	Sno	Cno	Grade
	0215121	李勇	男	20	CS		0215121	1	92
	0215122	刘晨	女	19	IS		0215121	2	85
	0215123	王敏	女	18	MA		0215126	2	88
	0215125	张立	男	18	IS		0215122	2	90
							0215122	3	80

➤ 用带EXISTS谓词的嵌套查询

SELECT Sname FROM Student

WHERE EXISTS

(SELECT * FROM SC

WHERE Sno=Student.Sno AND Cno= '2')

Sname
李勇
刘晨


带有EXISTS谓词的子查询示例

[例] 查询与“刘晨”在同一个系学习的学生学号和姓名。

- 1) 用EXISTS函数判断任一个学生t，其院系与刘晨院系是否相同
- 2) 从学生表中，搜索让上述逻辑式为真的元组

```
SELECT Sno, Sname  
FROM Student S1  
WHERE EXISTS
```

```
( SELECT *  
  FROM Student S2  
  WHERE S2.Sdept = S1.Sdept AND  
        S2.Sname = '刘晨' );
```



查询其他系中比计算机科学某一学生年龄小的学生姓名和年龄

```
SELECT Sname , Sage
FROM Student S1
WHERE EXISTS (SELECT * FROM Student S2
               WHERE Sdept = 'CS' AND S1.Sage < S2.Sage)
               AND Sdept <> 'CS' ;
```

注：S1是其他系学生元组变量，S2是计算机系元组变量。



带有EXISTS谓词的子查询(续)

[例] 查询没有选修1号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
  (SELECT *
   FROM SC
   WHERE Sno = Student.Sno AND Cno='1');
```

NOT EXISTS: 用于判断一个集合是否不为空。

- 若内层查询结果非空，则外层的WHERE子句返回假值
- 若内层查询结果为空，则外层的WHERE子句返回真值

外

Sno	Sname	Ssex	Sage	Sdept
0215121	李勇	男	20	CS
0215122	刘晨	女	19	IS
0215123	王敏	女	18	MA
0215125	张立	男	18	IS

内

Sno	Cno	Grade
0215121	1	92
0215121	2	85
0215126	2	88
0215122	2	90
0215122	3	80

带有EXISTS谓词的子查询(续)

[例] 查询没有选修1号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
  (SELECT *
   FROM SC
   WHERE Sno = Student.Sno AND Cno='1');
```

NOT EXISTS: 用于判断一个集合是否不为空。

- 若内层查询结果非空，则外层的WHERE子句返回假值
- 若内层查询结果为空，则外层的WHERE子句返回真值

Sno	Sname	Ssex	Sage	Sdept
0215121	李勇	男	20	CS
0215122	刘晨	女	19	IS
0215123	王敏	女	18	MA
0215125	张立	男	18	IS

Sno	Cno	Grade
0215121	1	92
0215121	2	85
0215126	2	88
0215122	2	90
0215122	3	80


```
select sno, sn from s
where not exist
( select *
  from sc
  where s.sno=sc.sno
    and g<60);
```

S

sno	sn	age	sex
s1	丁岩	19	M
s2	王爽	17	F
s3	李红	18	F
s4	赵立	21	M

查询所有课程成绩均及格的学生学号和姓名。

SC

sno	cno	G
s1	c1	79
s1	c3	85
s2	c1	83
s2	c2	56
s2	c3	90
s3	c1	95
s3	c2	80
s4	c1	85
s4	c2	55

带有EXISTS谓词的子查询(续)

- 不同形式的查询间的替换
 - 一些带EXISTS或NOT EXISTS谓词的子查询不能被其他形式的子查询等价替换
 - 所有带IN谓词、比较运算符、ANY和ALL谓词的子查询都能用带EXISTS谓词的子查询等价替换



用EXISTS/NOT EXISTS实现全称量词(难点)

SQL语言中没有全称量词 \forall (For all)

可以把带有全称量词的谓词转换为等价的带有存在量词的谓词:

$$(\forall x)P \equiv \neg (\exists x(\neg P))$$

即对于每个x 都满足P成立, 可以改写为: 不存在一个x使得P不成立



方法1:

用EXISTS/NOT EXISTS实现全称量词示例

[例] 查询选修了全部课程的学生姓名。

【分析】 “选修了全部课程” \Leftrightarrow “没有一门课他没有选”

SELECT Sname

FROM Student

WHERE NOT EXISTS //不存在学生t没选任一课程c

(SELECT * FROM Course

WHERE NOT EXISTS //学生t 选修了一门课程c， 为

(SELECT * FROM SC “假”，即：t没选课程c

WHERE Sno= Student.Sno //学生t 选修

AND Cno= Course.Cno) 了一门课程c

) ;

[例] 查询选修了全部课程的学生姓名。

- 方法2：用集函数

```
SELECT Sname  
FROM Student  
WHERE Sno IN  
( SELECT Sno  
  FROM SC  
   GROUP BY Sno  
   HAVING COUNT(*) =  
     (SELECT COUNT(*)  
      FROM Course));
```



用EXISTS/NOT EXISTS实现逻辑蕴含示例

[例] 查询至少选修了2号学生选修的全部课程的学生的学号。

【分析】

- 用逻辑蕴涵表达：查询学号为x的学生，对所有的课程y，只要2号学生选修了课程y，则x也选修了y。
- 形式化表示：
用P表示谓词 “2号学生选修了课程y”
用q表示谓词 “学生x选修了课程y”
则上述查询为: $(\forall y) p \rightarrow q$
- 利用谓词演算，做等价变换：
$$(\forall y)p \rightarrow q \equiv \neg \exists y(p \wedge \neg q)$$
- 变换后语义：不存在这样的课程y，2号学生选修了y，而学生x没有选。



- 题目要求：查询至少选修了2号学生选修的全部课程的学生学号，用NOT EXISTS谓词表示：

SELECT DISTINCT Sno

FROM SC X

WHERE NOT EXISTS //不存在一门课2号学生选了，学生t 没选

(SELECT * FROM SC Y

WHERE Sno = '2' AND

NOT EXISTS //学生t 没选2号学生选的一门课c

(SELECT * FROM SC Z

WHERE Z.Sno= X.Sno AND //学生t 选修了2号
Z.Cno= Y.Cno)); 学生选的一门课c



3.4 数据查询

3.4.1 单表查询

3.4.2 连接查询

3.4.3 嵌套查询

3.4.4 集合查询

3.4.5 基于派生表的查询



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



集合查询（续）

并操作

[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;
```

- 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。



集合查询（续）

方法二：

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



集合查询（续）

[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 ';
```



集合查询（续）

交操作

[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

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



集合查询（续）

- [例] 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;
```



集合查询（续）

[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

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



集合查询（续）

[例]实际上是查询既选修了课程1又选修了课程2
的学生

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



集合查询（续）

差操作

[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。

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



集合查询（续）

[例]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;
```



差操作（续）

[例] 查询学生姓名与教师姓名的差集

实际上是查询学校中未与教师同名的学生姓名

```
SELECT DISTINCT Sname
```

```
FROM Student
```

```
WHERE Sname NOT IN
```

```
  (SELECT Tname
```

```
   FROM Teacher);
```



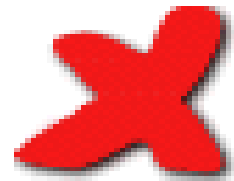
对集合操作结果的排序

- **ORDER BY**子句只能用于对最终查询结果排序，不能对中间结果排序
- 对集合操作结果排序时，**ORDER BY**子句中用数字指定排序属性



对集合操作结果的排序

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



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



基于派生表的查询

- 子查询还可以出现在**from**子句中。
- 派生表是在外部查询的**FROM**子句中定义的。派生表的存在范围为定义它的外部查询，只要外部查询一结束，派生表就不存在了。定义派生表的查询语句要写在一对圆括号内，后面跟着**AS**子句和派生表的名称。



[例] 找出每个超过他选修课程平均成绩的学生学号及课程号。

分析：

- 1) 用一个select块构造一个能够查询某个元组t的平均成绩的子函数
- 2) 从SC中搜索出满足其成绩大于该元组平均成绩的选课记录

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

相关子查询



例：找出每个学生超过他自己选修课程平均成绩的课程号。

SC:

Sno	Cno	Grade
200215121	1	92
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80

avg_sc

avg_sno	avg_grade
200215121	88
200215122	85

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.sno and sc.grade>=avg_sc.avg_grade



- 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缺省属性。
- [例3.60]查询所有选修了1号课程的学生姓名，可以用如下查询完成：

```
SELECT Sname  
FROM Student,  
(SELECT Sno FROM SC WHERE Cno='1 ') AS SC1  
WHERE Student.Sno = SC1.Sno;
```



3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 空值的处理

3.7 视图



3.5 数据更新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据



3.5.1 插入数据

- 两种插入数据方式
 1. 插入元组
 2. 插入子查询结果
- 可以一次插入多个元组



一、插入元组

- 语句格式

INSERT

INTO <表名> [(<属性列1>[, <属性列2 >...])]

VALUES (<常量1> [, <常量2>] ...)

- 功能：将新元组插入指定表中
- INTO子句
 - 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
 - 指定部分属性列
- VALUES子句
 - 提供的值必须与INTO子句匹配
 - 值的个数
 - 值的类型



插入元组（续）

[例1] 将一个新学生元组（学号：200215128；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 ('200215128', '陈冬', '男', 'IS', 18);



插入元组（续）

[例2] 将学生张成民的信息插入到Student表中。

```
INSERT  
INTO Student  
VALUES ('200215126', '张成民', '男', 18, 'CS');
```



插入元组（续）

[例3] 插入一条选课记录('200215128', '1 ')。

```
INSERT
```

```
INTO SC(Sno, Cno)
```

```
VALUES ( ' 200215128 ', ' 1 ' );
```

RDBMS将在新插入记录的Grade列上自动地赋空值。

或者：

```
INSERT
```

```
INTO SC
```

```
VALUES ( ' 200215128 ', ' 1 ', NULL);
```



二、插入子查询结果

- 语句格式

INSERT

INTO <表名> [(<属性列1> [, <属性列2>...])

子查询;

- 功能：将子查询结果插入指定表中
- INTO子句(与插入元组类似)
- 子查询
 - **SELECT**子句目标列必须与**INTO**子句匹配
 - 值的个数
 - 值的类型



插入子查询结果（续）

[例4] 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*/
```



插入子查询结果（续）

第二步：插入数据

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



插入子查询结果（续）

RDBMS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



3.5 数据更新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据



3.4.2 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>];

- SET子句

- 指定修改方式
- 要修改的列
- 修改后取值

- WHERE子句

- 指定要修改的元组
- 缺省表示要修改表中的所有元组

- 功能

- 修改指定表中满足WHERE子句条件的元组



修改数据（续）

- 三种修改方式
 1. 修改某一个元组的值
 2. 修改多个元组的值
 3. 带子查询的修改语句



1. 修改某一个元组的值

[例5] 将学生200215121的年龄改为22岁

```
UPDATE Student
```

```
SET Sage=22
```

```
WHERE Sno=' 200215121 ';
```



2. 修改多个元组的值

[例6] 将所有学生的年龄增加1岁

```
UPDATE Student
```

```
SET Sage= Sage+1;
```



3.5.2 修改数据

- 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7 将计算机科学系全体学生的成绩加10分。

```
UPDATE SC
```

```
SET grade=grade+10
```

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

处理过程类似于相关子查询



3.5.2 修改数据

- 解法2:

```
UPDATE sc
```

```
SET grade=grade+10
```

```
WHERE sno IN
```

```
  (SELECT sno
```

```
    FROM student
```

```
    WHERE sdept = 'CS');
```



修改数据（续）

RDBMS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
- 主码不允许修改
- 用户定义的完整性
 - NOT NULL约束
 - UNIQUE约束
 - 值域约束



3.5 数据更新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据



3.5.3 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



删除数据（续）

- 三种删除方式
 1. 删除某一个元组的值
 2. 删除多个元组的值
 3. 带子查询的删除语句



1. 删除某一个元组的值

[例8] 删除学号为200215128的学生记录。

DELETE

FROM Student

WHERE Sno= 200215128 ';



2. 删除多个元组的值

[例9] 删除所有的学生选课记录。

```
DELETE
```

```
FROM SC;
```



3.5.3 删除数据

三、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10 删除计算机科学系所有学生的选课记录

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



第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 空值的处理

3.7 视图



3.6 空值的处理

- 什么是空值

空值从技术上来说就是“未知的值”。但空值并不包括零、一个或者多个空格组成的字符串、以及零长度的字符串。

从实际应用中，空值说明还没有向数据库中输入相应的数据，或者某个特定的记录行不需要使用该列。在实际的操作中有下列几种情况可使得一列成为NULL。

- (1) 其值未知，如课程表中不明确具体的课程内容。
- (2) 其值不存在，如在学生表中某个学生由于没有参加考试，所以该学生的考试成绩为空值。
- (3) 由于某种原因不便于填写。



3.6 空值的处理

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);
```

或

```
Insert into sc(Sno,cno)  
Values('201215126','1');
```

例：将student表中学号为“201215200”的学生所属的系改为空值

```
Update student  
Set sdept=null  
Where sno=' 201215200';
```



3.6 空值的处理

2. 空值的判断

空值代表的是未知的值，所以并不是所有的空值都相等。例如，“**student**”表中有两个学生的年龄未知，但无法证明这两个学生的年龄相等。这样就不能用“=”运算符来检测空值。所以**SQL**引入了一个特殊的操作符**IS**来检测特殊值之间的等价性。

语法：WHERE Expression IS NULL

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;
```



3.6 空值的处理

3. 空值的约束条件

属性定义（或域定义）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. 空值的算术运算、比较运算和逻辑运算

- 如果null参与算术运算，则该算术表达式的值为null。
- 如果null参与比较运算，则结果可视为false。在SQL-92中可看成unknown。
- 如果null参与聚集运算，则除count(*)之外其它聚集函数都忽略null。



3.6 空值的处理

4. 空值的算术运算、比较运算和逻辑运算

例：选出选修1号课程的不及格学生以及缺考的学生学号

```
Select sno  
From sc  
Where grade<60 and cno='1'  
Union  
Select sno  
From sc  
Where grade is null and cno='1'
```



第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 空值的处理

3.7 视图



3.7 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

➤ 基于视图的操作

- 查询
- 删除
- 受限更新
- 定义基于该视图的新视图



3.7 视图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用



一、建立视图

- 语句格式

CREATE VIEW

<视图名> [(<列名> [, <列名>]...)]

AS <子查询>

[WITH CHECK OPTION];

- 组成视图的属性列名：全部省略或全部指定
- 子查询不允许含有**ORDER BY**子句和**DISTINCT**短语



建立视图（续）

- RDBMS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。
- 视图分类
 - 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。
 - 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。



建立视图（续）

[例1] 建立信息系学生的视图。

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



建立视图（续）

[例2]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;
```



建立视图（续）

对IS_Student视图的更新操作：

- 修改操作：自动加上Sdept= 'IS'的条件
- 删除操作：自动加上Sdept= 'IS'的条件
- 插入操作：自动检查Sdept属性值是否为'IS'
 - 如果不是，则拒绝该插入操作
 - 如果没有提供Sdept属性值，则自动定义Sdept为'IS'



建立视图（续）

- 基于多个基表的视图

[例3] 建立信息系选修了1号课程的学生视图。

```
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';
```



建立视图（续）

- 基于视图的视图

[例4] 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;
```



建立视图（续）

- 带表达式的视图

[例5] 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, 2022-Sage
FROM Student;
```

设置一些派生属性列, 也称为虚拟列--Sbirth

带表达式的视图必须明确定义组成视图的各个属性列名



建立视图（续）

- 分组视图

[例6] 将学生的学号及他的平均成绩定义为一个视图

假设SC表中“成绩”列Grade为数字型

```
CREAT VIEW S_G(Sno, Gavg)
```

```
AS
```

```
SELECT Sno, AVG(Grade)
```

```
FROM SC
```

```
GROUP BY Sno;
```



建立视图（续）

- 不指定属性列

[例7]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 *
FROM Student
WHERE Ssex='女' ;
```

缺点：

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。



二、删除视图

- 语句的格式:

DROP VIEW <视图名>;

- 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



删除视图(续)

[例8] 删除视图BT_S: DROP VIEW BT_S;

删除视图IS_S1: DROP VIEW IS_S1;

➤拒绝执行

➤级联删除:

DROP VIEW IS_S1 CASCADE;



3.7 视图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用



3.7.2 查询视图

- 用户角度：查询视图与查询基本表相同
- RDBMS实现视图查询的方法
 - 视图消解法（View Resolution）
 - 进行有效性检查
 - 转换成等价的对基本表的查询
 - 执行修正后的查询



查询视图（续）

[例9] 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;
```

IS_Student视图的定义 (参见视图定义例1)

视图消解转换后的查询语句为:

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



查询视图（续）

[例10] 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';
```



查询视图（续）

- 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。



查询视图（续）

[例11]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;
```



查询转换

错误:

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

Where语句不能用聚集函数作为
条件表达式

正确:

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



3.7 视图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用



更新视图（续）

[例12] 将信息系学生视图IS_Student中学号200215122的学生姓名改为“刘辰”。

```
UPDATE IS_Student  
SET Sname= '刘辰'  
WHERE Sno= ' 200215122 ';
```

转换后的语句：

```
UPDATE Student  
SET Sname= '刘辰'  
WHERE Sno= ' 200215122 ' AND Sdept= 'IS';
```



更新视图（续）

[例13] 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('200215129', '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' );
```



更新视图（续）

[例14]删除信息系学生视图IS_Student中学号为200215129的记录

```
DELETE  
FROM IS_Student  
WHERE Sno= ' 200215129 ';
```

转换为对基本表的更新:

```
DELETE  
FROM Student  
WHERE Sno= ' 200215129 ' AND Sdept= 'IS';
```



更新视图（续）

- 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

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



更新视图（续）

- 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



3.7 视图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用



3.7.4 视图的作用

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



1. 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



2. 视图使用户能以多种角度看待同一数据

- 视图机制能使不同用户以不同方式看待同一数据，适应数据库共享的需要



3.视图对重构数据库提供了一定程度的逻辑独立性

例：数据库逻辑结构发生改变

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)



3.视图对重构数据库提供了一定程度的逻辑独立性

通过建立一个视图**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;
```

使用户的外模式保持不变，从而对原**Student**表的
查询程序不必修改



4. 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
- 通过**WITH CHECK OPTION**对关键数据定义操作时间限制



建立视图（续）

建立1号课程的选课视图，并要求透过该视图进行的更新操作只涉及1号课程，同时对该视图的任何操作只能在工作时间进行。

```
CREATE VIEW IS_SC
AS
SELECT Sno, Cno, Grade
FROM SC
WHERE Cno= '1'
AND TO_CHAR(SYSDATE,'HH24') BETWEEN 9 AND 17
AND TO_CHAR(SYSDATE,'D') BETWEEN 1 AND 5
WITH CHECK OPTION;
```



5. 适当的利用视图可以更清晰的表达查询

例如：对每个同学找出他获得最高成绩的课程号。

解题思路：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;

然后再用如下查询语句完成查询：

Select sc.sno,cno

From sc,vmgrade

Where sc.sno=vmgrade.sno and sc.grade=vmgrade.mgrade;

