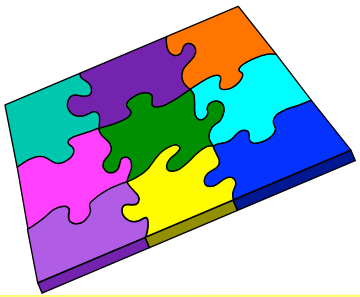
数 据 库 系 统 概 论
An Introduction to Database Systems

第三章
关系数据库标准语言SQL

第三章　关系数据库标准语言SQL

本章主要内容

- 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
- 小结



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
- 3.4.6 Select语句的一般形式

➢连接查询：同时涉及多个表的查询

➢连接条件或连接谓词：用来连接两个表的条件

一般格式：

[<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>

[<表名1>.]<列名1>

BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

等值与非等值连接查询

若“表名1”与“表名2”是同一张表，则为自身连接

OUT JOIN: 外连接

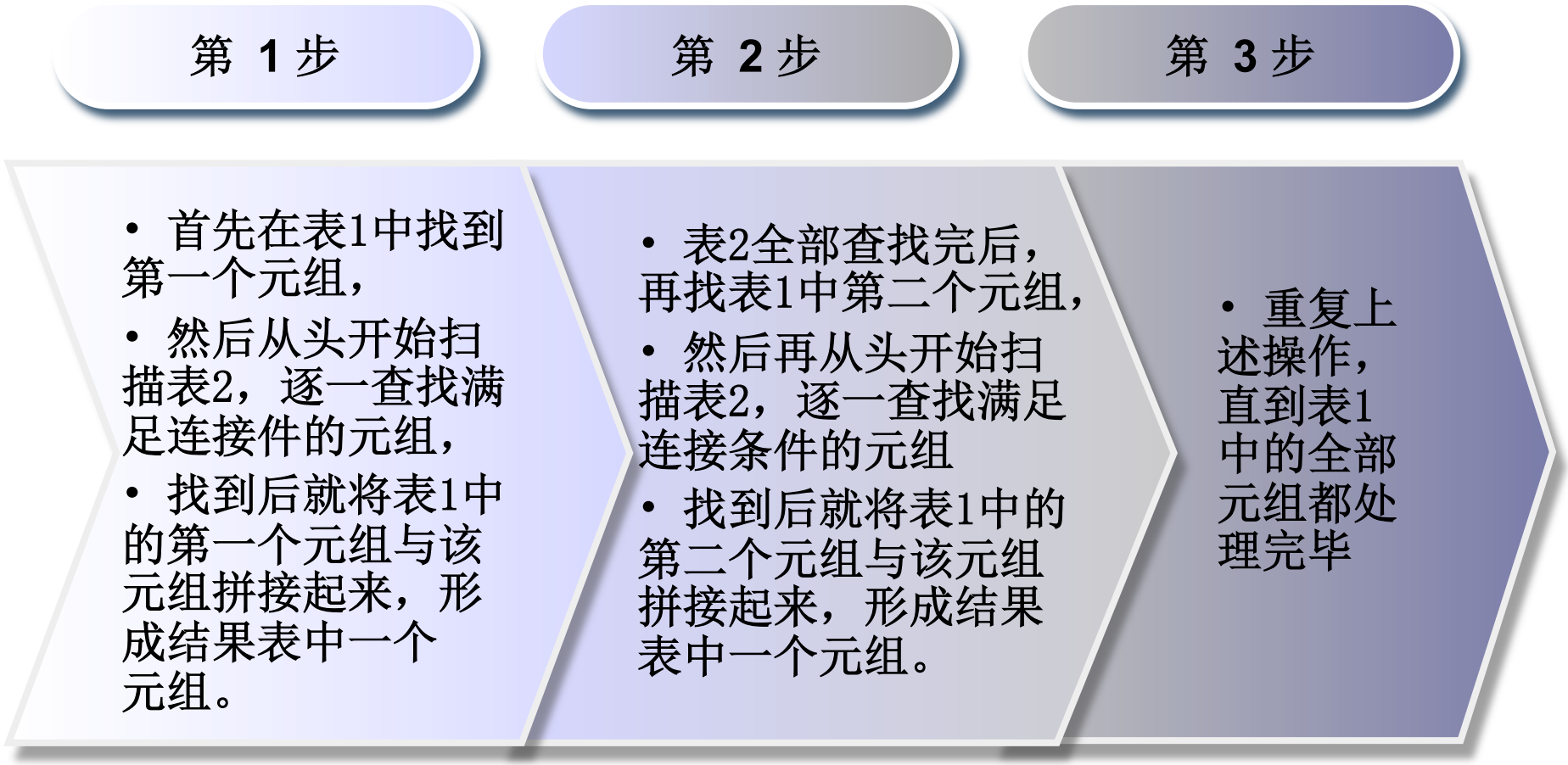
若还包括其它连接条件，则为复合条件连接

➢连接字段：连接谓词中的列名称

➢连接条件中的各连接字段类型必须是可比的，但名字不必是相同的 不能是字符型和数字型进行比较

3.4.2 连接查询

- 连接操作的执行过程
  - 嵌套循环法(NESTED-LOOP)



3.4.2 连接查询

连接操作的执行过程

常用于=连接

排序合并法(SORT-MERGE)

1. 首先按连接属性对表1和表2排序
2. • 对表1的第一个元组，从头开始扫描表2，顺序查找满足连接条件的元组
   • 找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。
   • 当遇到表2中第一条大于表1连接字段值的元组时，对表2的查询不再继续
3. • 找到表1的第二条元组，
   • 然后从刚才的中断点处继续顺序扫描表2，查找满足连接条件的元组，
   • 找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组
   • 直接遇到表2中大于表1连接字段值的元组时，对表2的查询不再继续
4. 重复上述操作，直到表1或表2中的全部元组都处理完毕为止

大多数查询都是根据我们定义的参照完整性来完成不同表之间的连接

3.4.2 连接查询

➲ 索引连接(INDEX-JOIN)

大多数表在建立的时候往往会直接生产索引

- 对表2按连接字段建立索引
- 对表1中的每个元组，依次根据其连接字段值查询表2的索引，从中找到满足条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组

一、等值与非等值连接查询

➲ 等值连接：　　连接运算符为=

[例33] 查询每个学生及其选修课程的情况

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

查询结果：

Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
200215121	李勇	男	20	CS	200215121	1	92
200215121	李勇	男	20	CS	200215121	2	85
200215121	李勇	男	20	CS	200215121	3	88
200215122	刘晨	女	19	CS	200215122	2	90
200215122	刘晨	女	19	CS	200215122	3	80

一、等值与非等值连接查询

➲ 自然连接：

[例34] 对[例33]用自然连接完成。

```
SELECT  Student.Sno，Sname，Ssex，Sage，Sdept，Cno，Grade
FROM    Student，SC
WHERE  Student.Sno = SC.Sno；
```

二、自身连接

- 自身连接：一个表与其自己进行连接
  - 需要给表起别名以示区别
  - 由于所有属性名都是同名属性，因此必须使用别名前缀

[例35]查询每一门课的间接先修课（即先修课的先修课）

```
SELECT FIRST.Cno，SECOND.Cpno
FROM Course FIRST，Course SECOND
WHERE FIRST.Cpno = SECOND.Cno；
```

查询结果：

Cno	Pcno
1	7
3	5
5	6

二、自身连接

FIRST表（Course表）

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

SECOND表（Course表）

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

```
SELECT FIRST.Cno，SECOND.Cpno
FROM Course FIRST，Course SECOND
WHERE FIRST.Cpno = SECOND.Cno
```

三、外连接

- 外连接与普通连接的区别
  - 普通连接操作只输出满足连接条件的元组
  - 外连接操作以指定表为连接主体，将主体表中不满足连接条件的元组一并输出
- 左外连接
  - 列出左边关系（如本例**Student**）中所有的元组
- 右外连接
  - 列出右边关系中所有的元组

三、外连接

[例 36] 改写[例33]

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

执行结果：

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
200215121	李勇	男	20	CS	1	92
200215121	李勇	男	20	CS	2	85
200215121	李勇	男	20	CS	3	88
200215122	刘晨	女	19	CS	2	90
200215122	刘晨	女	19	CS	3	80
200215123	王敏	女	18	MA	NULL	NULL
200215125	张立	男	19	IS	NULL	NULL

四、复合条件连接

- 复合条件连接：WHERE子句中含多个连接条件

[例37]查询选修2号课程且成绩在90分以上的所有学生的学号和姓名

```
SELECT Student.Sno, Sname
FROM   Student, SC
WHERE Student.Sno = SC.Sno        /* 连接谓词*/
AND   SC.Cno= '2' AND SC.Grade > 90;
                                  /* 其他限定条件 */
```

四、复合条件连接

[例38]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   /*多表连接*/
WHERE Student.Sno = SC.Sno
        and SC.Cno = Course.Cno；
```

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
- 3.4.6 Select语句的一般形式

3.4.3 嵌套查询

嵌套查询概述

- 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

```
SELECT Sname                          /*外层查询/父查询*/
FROM Student
WHERE Sno IN
        (SELECT Sno                   /*内层查询/子查询*/
         FROM SC
         WHERE Cno= ' 2 ');
```

3.4.3 嵌套查询

- 嵌套查询概述
  - 子查询的限制
    - 不能使用**ORDER BY**子句
  - 层层嵌套方式反映了 **SQL**语言的结构化
  - 有些嵌套查询可以用连接运算替代

3.4.3 嵌套查询

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。

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

3.4.3 嵌套查询

- 一、 带有IN谓词的子查询
- 二、 带有比较运算符的子查询
- 三、 带有ANY（SOME）或ALL谓词的子查询
- 四、 带有EXISTS谓词的子查询

一、带有IN谓词的子查询

[例39] 查询与“刘晨”在同一个系学习的学生。

此查询要求可以分步来完成

① 确定“刘晨”所在系名

```
SELECT  Sdept
FROM    Student
WHERE  Sname= ' 刘晨 ';
```

结果为： CS

② 查找所有在CS系学习的学生。

```
SELECT  Sno，Sname，Sdept
FROM    Student
WHERE  Sdept= ' CS ';
```

结果为：

Sno	Sname	Sdept
200215121	李勇	CS
200215122	刘晨	CS

3.4.3 嵌套查询

一、带有IN谓词的子查询

将第一步查询嵌入到第二步查询的条件中

```
SELECT Sno，Sname，Sdept
FROM Student
WHERE Sdept  IN
      (SELECT Sdept
       FROM Student
       WHERE Sname= ‘ 刘晨 ’ );
```

可以用in更加不容易出错

此查询为

不相关子查询

用自身连接完成[例39]查询要求

```
SELECT S1.Sno，S1.Sname，S1.Sdept
FROM   Student S1，Student S2
WHERE S1.Sdept = S2.Sdept
      AND S2.Sname = '刘晨';
```

3.4.3 嵌套查询

一、带有IN谓词的子查询

[例40]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关系中找出选修了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

一、带有IN谓词的子查询

用连接查询实现[例40]

```
SELECT Sno，Sname
FROM   Student，SC，Course
WHERE  Student.Sno = SC.Sno
    AND  SC.Cno = Course.Cno
    AND   Course.Cname='信息系统'；
```

二、带有比较运算符的子查询

- 当能确切知道内层查询返回单值时，可用比较运算符
  - 比较运算符包括

（>，<，=，>=，<=，!=或< >）

- 与ANY或ALL谓词配合使用

二、带有比较运算符的子查询

例：假设一个学生只可能在一个系学习，并且必须属于一个系，则在[例39]可以用 = 代替IN：

```
SELECT Sno，Sname，Sdept
FROM   Student
WHERE Sdept  =
  (SELECT Sdept
    FROM   Student
    WHERE Sname= ‘刘晨’);
```

错误的例子：

```
SELECT  Sno，Sname，Sdept
FROM    Student
WHERE ( SELECT Sdept
          FROM Student
          WHERE Sname= ‘刘晨’ )
          = Sdept;
```

[注]: *子查询一定要跟在比较符之后*

二、带有比较运算符的子查询

［例41］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);
```

相关子查询

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）

三、带有ANY（SOME）或ALL谓词的子查询

需要配合使用比较运算符

> ANY	大于子查询结果中的某个值
> ALL	大于子查询结果中的所有值
< ANY	小于子查询结果中的某个值
< ALL	小于子查询结果中的所有值
>= ANY	大于等于子查询结果中的某个值
>= ALL	大于等于子查询结果中的所有值
<= ANY	小于等于子查询结果中的某个值
<= ALL	小于等于子查询结果中的所有值
= ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值（通常没有实际意义）
!=（或<>） ANY	不等于子查询结果中的某个值
!=（或<>） ALL	不等于子查询结果中的任何一个值

谓词语义

ANY：某一个值

ALL：所有值

三、带有ANY（SOME）或ALL谓词的子查询

[例42] 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
WHERE Sage < ANY
        (SELECT  Sage
         FROM    Student
         WHERE Sdept= ' CS ')
         AND Sdept <> 'CS ' ;        /*父查询块中的条件 */
```

执行过程：

1.RDBMS执行此查询时，首先处理子查询，找出CS系中所有学生的年龄，构成一个集合(20，19)

2. 处理父查询，找所有不是CS系且年龄小于20 或19的学生

结果：

Sname	Sage
王敏	18
张立	19

子查询语句一定要跟在比较运算符后面

三、带有ANY（SOME）或ALL谓词的子查询

用聚集函数实现[例42]

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

三、带有ANY（SOME）或ALL谓词的子查询

[例43] 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 ';
```

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 ';
```

三、带有ANY（SOME）或ALL谓词的子查询

表3.5　ANY（或SOME），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

四、带有EXISTS谓词的子查询

- 1. EXISTS谓词
  - 存在量词∃
  - 带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”。
    - 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
  - 由EXISTS引出的子查询，其目标列表达式通常都用* ，因为带EXISTS的子查询只返回真值或假值，给出列名无实际意义
- 2. NOT EXISTS谓词
    - 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

四、带有EXISTS谓词的子查询

[例44]查询所有选修了1号课程的学生姓名。

思路分析：

- 本查询涉及Student和SC关系
- 在Student中依次取每个元组的Sno值，用此值去检查SC关系
- 若SC中存在这样的元组，其Sno值等于此Student.Sno值，并且其Cno= '1'，则取此Student.Sname送入结果关系

四、带有EXISTS谓词的子查询

- 用嵌套查询 等价 用连接运算

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

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

相关嵌套，面对同一个查询要求可以用不同的查询语句

四、带有EXISTS谓词的子查询

［例45］ 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');
```

此查询用连接运算就不太容易实现了！

四、带有EXISTS谓词的子查询

- 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

四、带有EXISTS谓词的子查询

例：[例39]查询与“刘晨”在同一个系学习的学生。

可以用带EXISTS谓词的子查询替换：

```
SELECT Sno，Sname，Sdept
FROM Student S1
WHERE EXISTS
        (SELECT *
        FROM Student S2
        WHERE S2.Sdept = S1.Sdept AND
              S2.Sname = ‘刘晨’);
```

四、带有EXISTS谓词的子查询

- 用EXISTS/NOT EXISTS实现全称量词(难点)
  - SQL语言中没有全称量词∀ （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：

$$(\forall x)P \equiv \neg (\exists x(\neg P))$$ 不存在x使得p不成立

四、带有EXISTS谓词的子查询

［例46］查询选修了全部课程的学生姓名。

元祖关系演算——用两种量词的检索 元组关系演算这部分不做要求

RANGE Course CX

SC SCX

GET W (Student.Sname):

∀ CX ∃SCX (SCX.Sno=Student.Sno∧

SCX.Cno=CX.Cno) sno和cno存在

等 价

¬∃CX¬ ∃SCX (SCX.Sno=Student.Sno∧

SCX.Cno=CX.Cno)

四、带有EXISTS谓词的子查询

［例46］查询选修了全部课程的学生姓名。

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

四、带有EXISTS谓词的子查询

- 用EXISTS/NOT EXISTS实现逻辑蕴函(难点)
  - SQL语言中没有蕴函(Implication)逻辑运算
  - 可以利用谓词演算将逻辑蕴函谓词等价转换为：

$$p \rightarrow q \equiv \neg p \vee q$$

p为前件，q为后件，其真值指派应该是：

p	q	p→q	
T	T	T	1
T	F	F	2
F	T	T	3
F	F	T	4

四、带有EXISTS谓词的子查询

[例47]查询至少选修了学生200215122选修的全部课程的学生号码。

解题思路：

- 用逻辑蕴函表达：查询学号为x的学生，对所有的课程y，只要200215122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 “学生200215122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: $(\forall y)\ p \rightarrow q$

3.4.3 嵌套查询

一、检索操作

[例47] 查询至少选修了200215122学生所选全部课程的学生学号

RANGE Couse CX

SC SCX

SC SCY

GET W (Student.Sno) :

$\forall$ CX($\exists$SCX(SCX.Sno='200215122'$\wedge$SCX.Cno=CX.Cno)

$\Longrightarrow$ $\exists$SCY(SCY.Sno=Student.Sno$\wedge$SCY.Cno= CX.Cno))

四、带有EXISTS谓词的子查询

- 等价变换：

$$(\forall y)p \rightarrow q \equiv \neg (\exists y (\neg(p \rightarrow q)))$$
$$\equiv \neg (\exists y (\neg(\neg p \vee q)))$$
$$\equiv \neg \exists y(p \wedge \neg q)$$

- 变换后语义：不存在这样的课程y，学生200215122选修了y，而学生x没有选。

四、带有EXISTS谓词的子查询

- 用NOT EXISTS谓词表示：

```
SELECT DISTINCT Sno
FROM SC SCX
WHERE NOT EXISTS
        (SELECT *
         FROM SC SCY
         WHERE SCY.Sno = ' 200215122 '  AND
                 NOT EXISTS
                 (SELECT *
                  FROM SC SCZ
                  WHERE SCZ.Sno=SCX.Sno AND
                          SCZ.Cno=SCY.Cno));
```

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
- 3.4.6 Select语句的一般形式

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48] 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

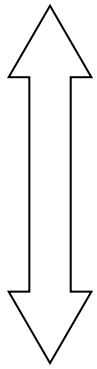
[例49] 查询选修了课程1或者选修了课程2的学生。

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

3.4.4 集合查询

[例50] 查询计算机科学系的学生与年龄不大于19岁的学生的交

```
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;
```

3.4.4 集合查询

[例51] 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 ';
```

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 ')
```

3.4.4 集合查询

[例52] 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;
```

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;
```

3.4 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*
- 3.4.6 **Select语句的一般形式**

3.4.5 基于派生表的查询

［例41］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);
```

```
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
```

3.4.5 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，此时，通过子查询生成的临时派生表成为主查询的查询对象。
- 若子查询中没有聚集函数，派生表可不指定属性列，子查询**SELECT**子句后面的列名为其默认属性。

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

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
- 3.4.6 *Select语句的一般形式*

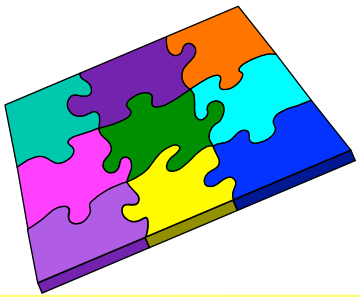
3.4.5 SELECT语句的一般格式

SELECT [ALL|DISTINCT]
<目标列表达式> [别名] [，<目标列表达式> [别名]] …
FROM <表名或视图名> [别名]
[，<表名或视图名> [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]

第三章　关系数据库标准语言SQL

本章主要内容

- 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
- 小结



3.5 数据更新

- 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

3.5.1 插入数据

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

3.5.1 插入数据

一、插入元组

［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);
```

［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');
```

一、插入元组

［例3］ 插入一条选课记录('200215128'，'1 ')。

```
 INSERT
INTO SC(Sno，Cno)
VALUES (‘ 200215128 ’，  ‘ 1 ’);
```

或者：

```
 INSERT
INTO SC
VALUES (' 200215128 '，' 1 '，NULL);
```

RDBMS将在新插入记录的Grade列上自动地赋空值。

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

二、插入子查询结果

[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*/
```

第二步：插入数据

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

二、插入子查询结果

- **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

插入数据

修改数据

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>];
```

功能——修改指定表中满足WHERE子句条件的元组

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

3.4.2 修改数据

- 三种修改方式

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. 带子查询的修改语句

[例7] 将计算机科学系全体学生的成绩置零。

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

3.4.2 修改数据

- **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

插入数据

修改数据

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

3.5.3 删除数据

- 三种删除方式

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. 带子查询的删除语句

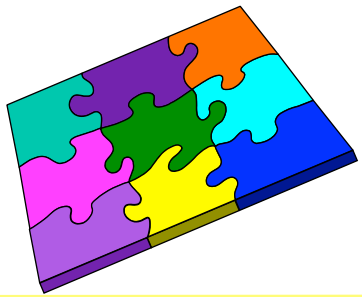
[例10] 删除计算机科学系所有学生的选课记录。

```
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

本章主要内容

- 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
- 小结



3.6 空值的处理

- 空值就是“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

3.6.1 空值的产生

- 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。

- 空值的产生

[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*/
```

3.6.1 空值的产生

[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';
```

3.6.2 空值的判断

- 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;
```

3.6.3 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

3.6.4 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

3.6.4 空值的算术运算、比较运算和逻辑运算

表3.8　逻辑运算符真值表

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表示TRUE，F表示FALSE，U表示UNKNOWN

3.6.4 空值的算术运算、比较运算和逻辑运算

[例3.82] 找出选修1号课程的不及格的学生。

```
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值为null。

3.6.4 空值的算术运算、比较运算和逻辑运算

[例 3.83] 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 Grade < 60 AND Cno='1'
UNION
SELECT Sno
FROM SC
WHERE Grade IS NULL AND Cno='1'
```

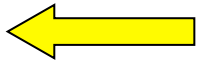
或者

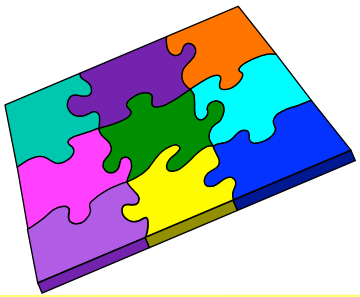
```
SELECT Sno
FROM SC
WHERE Cno='1' AND（Grade<60 OR Grade IS NULL）;
```

第三章　关系数据库标准语言SQL

本章主要内容

- 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
- 小结





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

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
- 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。

[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'
WITH CHECK OPTION;
```

且要求：

对IS_Student视图进行更新操作时仍需保证该视图只有信息系的学生

- 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'

一、建立视图

- 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      Student.Sno=SC.Sno
          AND   Sdept= 'IS'
          AND   SC.Cno= '1';
```

一、建立视图

- 基于视图的视图

[例4] 建立信息系统选修了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；
```

- 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，2016-Sage
FROM Student；
```

一、建立视图

➲ 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为数值型

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

一、建立视图

- 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

二、删除视图

［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

3.7.2 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

3.7.2 查询视图

[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)

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

视图消解转换后的查询语句为：

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

3.7.2 查询视图

[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';
```

视图消解转换后的查询语句为：

```
SELECT  Student.Sno，Sname
FROM    Student，SC
WHERE  Student.Sno =SC.Sno AND SC.Cno= '1 '
          AND Student.Sdept= 'IS' ；
```

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。

[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视图的定义：

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

3.7.2 查询视图

➲ 查询转换

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 视图

3.7.3 更新视图

[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';
```

3.7.3 更新视图

[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' );
```

3.7.3 更新视图

[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';
```

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
  - 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

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。

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 视图

3.7.4 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
- 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

习题

建立School 数据库，并对数据库进行查询操作

其中，School数据库包括四张表格，分别为：

- 表STUDENTS (sid，sname，email，grade)；
- 表TEACHERS（tid，tname，email，salary）；
- 表COURSES（cid，cname，hour）；
- 表CHOICES（no，sid，tid，cid，score）。

在该数据库中，存在的关系为：
学生可以选择课程，一门课程对应一位教师。

- 请按要求对School数据库进行查询操作：
  - 查询所有选修记录的课程号
  - 查询所有教师的编号及选修其课程的学生的平均成绩，按平均成绩的降序排列
  - 求出至少被两名学生选修的课程编号
  - 查询所有选了database的学生的编号
  - 找出每门选修课程成绩最好的选课记录
  - 查询所有选修编号10001的课程的学生姓名
  - 查询选择课程C++或选择课程Java的学生的编号和姓名